MERGING - FUNCTIONS

24710-2/8508

Printed in South Africa

SCHOOL OF COMPUTER TRAINING

PROGRAMMING IN BASIC
STUDY UNIT 10

MERGING — FUNCTIONS

Edition 2

STUDY UNIT 10

YOUR LEARNING OBJECTIVES

WHEN YOU COMPLETE THIS UNIT, YOU WILL BE ABLE TO:

STUDY UNIT 10

MERGING—FUNCTIONS

DO YOU KNOW?

- **How menus can make software "user-friendly"?**
- **How a bubble sort works?**
- **What stub testing means?**

HUMANIZING THE PROGRAMS

Do you remember how "whelmed" you felt the first time you visited a computer store? How about the first attempts you made to input data on a keyboard? Have you experienced the frustration of putting in a program and data, but not being able to get it out? Have you ever "lost" your program?

There is no doubt about it. Computer "lingo" and the act of learning how to use a computer are pretty intimidating to the person just starting out. By now, of course, you have gotten over "computer shock" and learned how to make the computer be your own personal genie.

With this experience and knowledge comes a certain amount of responsibility. Being a programmer means that you will be devising programs for users who may not know very much about computers, programs, data or applications. All they know is that everybody is using one, and they have been told that a computer can save them plenty of money and time.

FIGURE 1—Soon, nearly everyone will have some form of programmable computer for work and pleasure. It is the programmer's responsibility to provide software which is easily used and totally reliable.

What sort of program will you write for a customer who is willing to hire you in hopes that you will have answers? Once you realize how little your client knows about computers and programs, you do have a responsibility to *educate* and *assist* this user.

The first and most essential task is to find out precisely how a computer will assist the client toward greater efficiency, profits, and control. Once you really understand the job requirements, you can set about creating programs which will actually accomplish two objectives at the same time. They will

- meet the specified job requirements, and
- educate the program user in how to obtain reliable results.

Meeting these objectives is the ideal you—the programmer—should always strive to achieve. With your skills and experience, you should be able to provide customers with programs which they can use without much of the initial frustration and "computer shock" you first experienced.

There is no secret knack or knowledge required to produce programs which a beginner can understand. It does require patience, a true understanding about what the user does not know, and a bit of trial and error practice on your part.

This doesn't mean that you have to stay with "simple" programming. Rather, it means you have to do some careful, step-by-step guiding of your pupil through a new and often confusing process. Always keep in mind what you experienced when you set about programming for others. And, keep the user constantly in mind. After all, you don't want to lose a client halfway through a job because he or she got intimidated and "whelmed".

This Study Unit will show you how you can teach the novice step-by-step procedures in setting up and using programs. And if you have done your programming tasks well, your program and the computer will actually teach the client how to take command of the genie! Once you have achieved that, you have really passed your final test as a successful programmer. Let's see how to do it!

MENUS AND USER-FRIENDLINESS

The technique of storing and maintaining files of data is only useful if the information is easy to retrieve and update. While the programmer may have to spend considerable time to produce a "bug-free" program, the user will only be impressed by simplicity, accuracy, and speed. Very often, the design and coding of a program will be more complex, internally, in order to make it easy to use.

The term "user-friendly" expresses this seeming paradox. While this term has often been used as a mere marketing tool to sell software, the more commercially successful programs are truly easy to use. After all, we are the programmers; we should assume only that the users of our applications can read, follow instructions and have some knowledge of the keyboard. As soon as the program is executed, all technical knowledge should be invisible to the user.

HOW TO ACHIEVE "USER-FRIENDLINESS"

1. **Printing clear and simple instructions for each prompt.**
2. **Limiting the amount of data the user must enter from the keyboard.**
3. **Displaying only the output that the user has requested.**
4. **Allowing the users to "change their minds"—that is, to cancel an entry.**
5. **Printing simple error messages which show the user what data they entered was incorrect, without "blowing up" the program.**

FIGURE 2—"User-friendliness" means making programs as easy as possible for users. The simpler the instructions, the easier it will be for operators to learn how to apply programs successfully.

One application may offer the user several different functions. For example, we can code a program which will allow the user the opportunity to:

- ☐ Create a master file.
- ☐ Update or add to that file at a later date.
- ☐ Rearrange the data entered.
- ☐ Display the data.

As soon as the program begins to execute, the user will have to decide which of these functions to choose. A "menu" can be employed to make this choice user-friendly.

A menu is nothing more than a display of a list of options and a prompt. This is closely analogous to a restaurant menu, which is a list of food items available to the user (or, in this case, the consumer). But, whereas, in a restaurant your choice is relayed to a waiter, in a program, the "choice" is the response the user makes to a prompt. Let's see how this works when designing and coding a program for a business executive. The busy executive is *always* making choices—especially when setting appointments. A "flexible" program for maintaining appointments could be an excellent timesaver.

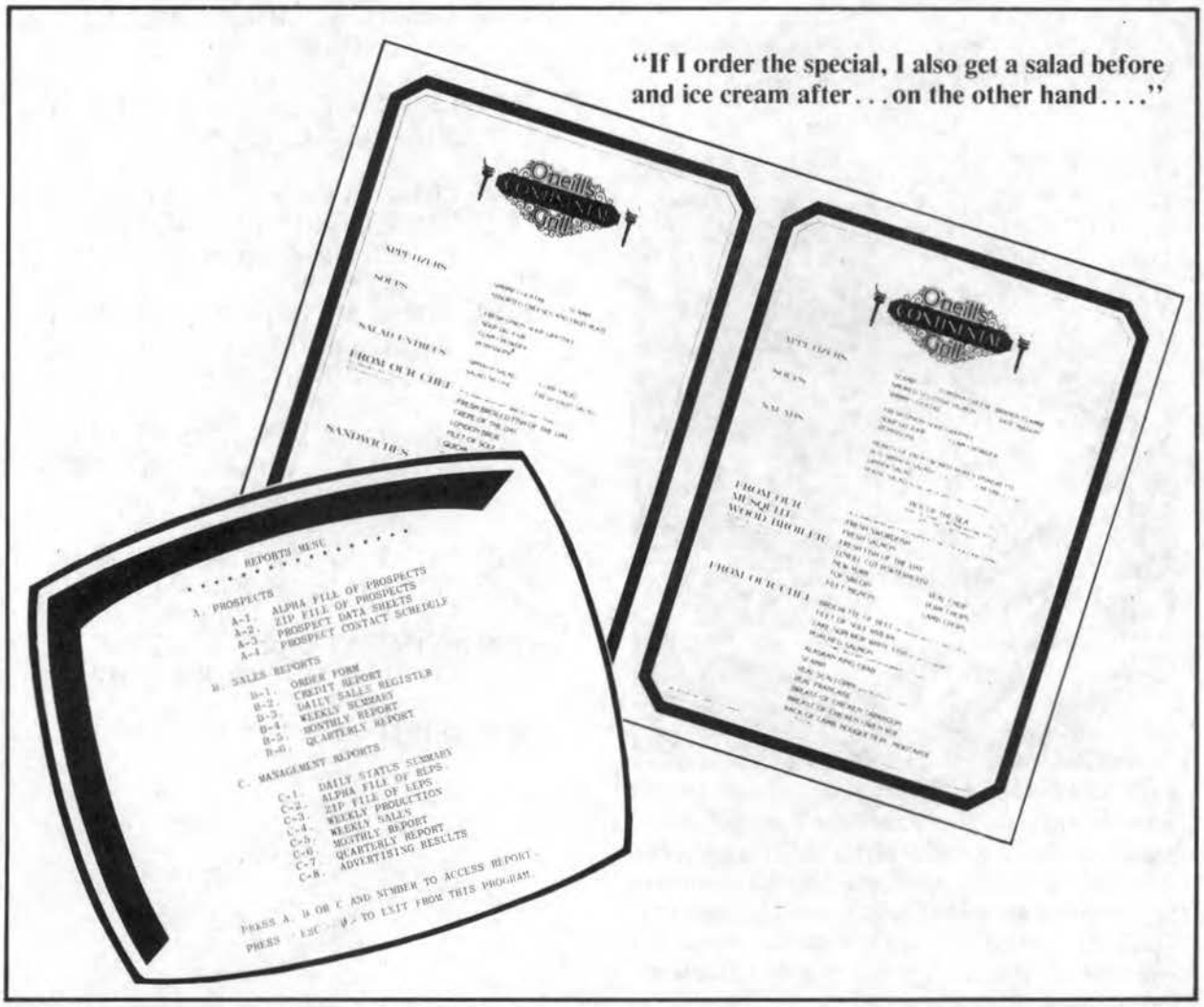


FIGURE 3—There is very little difference between a program menu and a menu found in a restaurant. Both offer "choices", depending upon the requirements of the customer.

CREATING AN APPOINTMENT CALENDAR PROGRAM

This program will store a list of dates, times, locations and names of various appointments the executive's secretary wants to keep on a month-by-month basis. We will allow the user to enter the data, update or cancel appointments, display meetings on a daily basis, or to display the dates on which a specified person must be met.

This program will require several subsections—one for each of the various options. At the beginning of the run, a menu listing these options will be displayed.

Before we begin coding the subsections, let's see how the menu will appear.

FIGURE 4—The secretary is usually the person who maintains the executive's appointment calendar. The calendar alerts the secretary and executive to tasks which must be accomplished before meeting with a client, too. A computerized calendar can accommodate the day-to-day changes so that complete accuracy is always maintained.

The coding of this menu will be no more than a series of print statements with a prompt at the end. Since this menu will be displayed several times during the run (the menu will be redisplayed after each subsection is executed as well as at the start), let's code it as a subroutine:

```
10 GOSUB 8000

20 STOP

8000 REM DISPLAY MENU

8010 CLS

8020 PRINT AT 0,4; "APPOINTMENTS
     CALENDAR"

8030 PRINT AT 2,0; "NUMBER"; AT
     2,16; "FUNCTION"

8040 PRINT AT 4,3; "1"; AT 4,12;
     "ENTER APPOINTMENTS"; AT
     5,13; "FOR A MONTH"

8050 PRINT AT 7,3; "2"; AT 7,12;
     "CHANGE OR REMOVE"; AT
     8,13; "APPOINTMENTS"

8060 PRINT AT 10,3; "3"; AT 10,12;
     "DISPLAY APPOINTMENTS"; AT
     11,13; "BY DAY OR NAME"

8070 PRINT AT 13,3; "4"; AT 13,12;
     "END THE PROGRAM"

8080 PRINT AT 21,0; "ENTER THE
     NUMBER OF YOUR CHOICE"

8090 INPUT A$
```

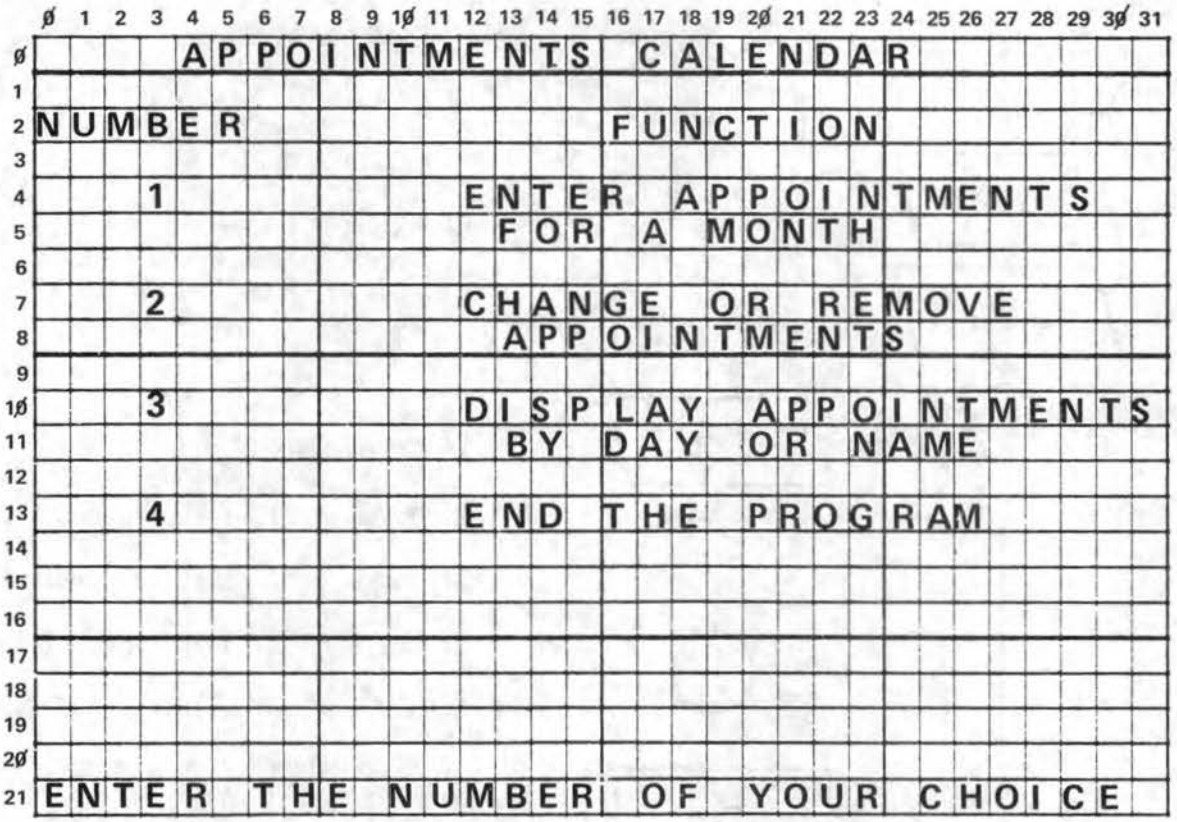
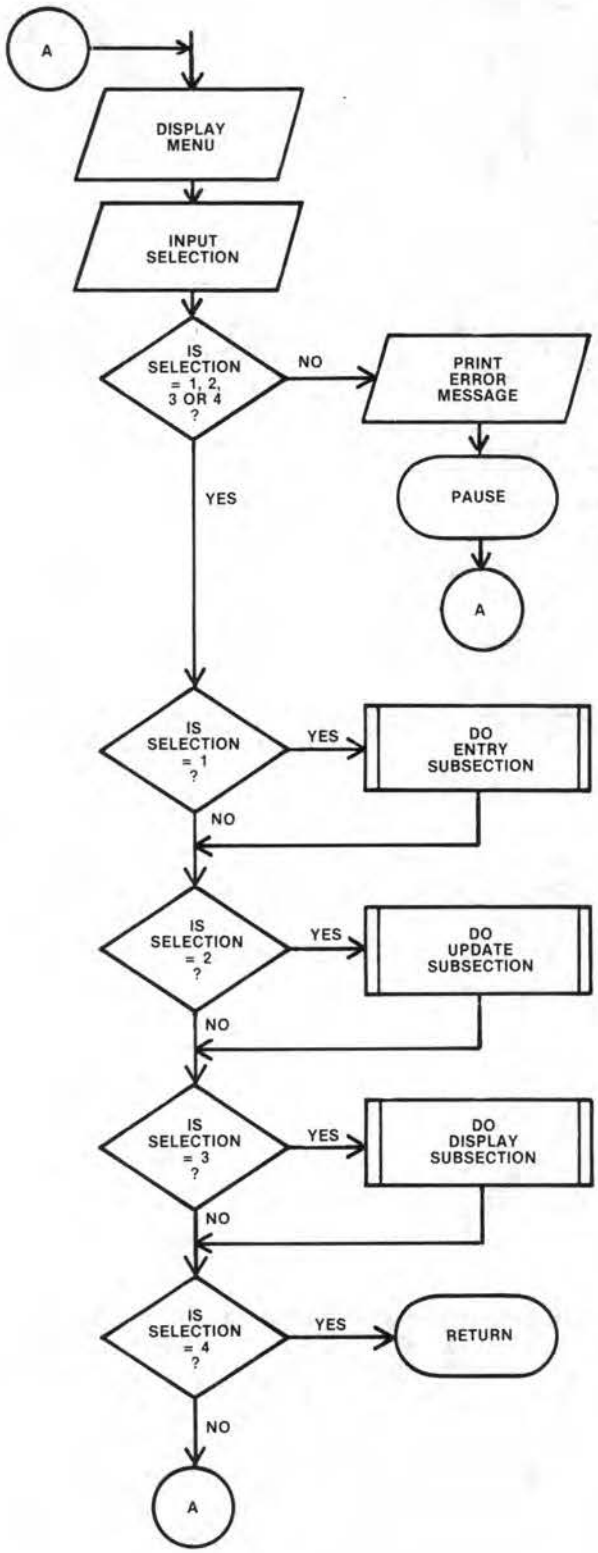


FIGURE 5—Sketching out menus on a grid helps in identifying the best visual arrangement. It is also quite easy to enter correct line and column numbers when the layout—such as this menu—is complete.

Enter and run this short program. You will see the menu displayed just as the user will see it. Note that the user need only enter a single number in response to the prompt and that the meaning of each selection is clearly displayed on the screen.

But we still do not have a "menu-driven" program since whatever the response of the user to the prompt, the program will end. What we have to do now is to test the response and branch to different sections of our program. Also, we must ensure that the user has given a valid response (1, 2, 3 or 4). If not, an error message should be displayed and the user given another opportunity to select a proper number.

A flowchart of this might be designed as follows:



This design could then be coded as given below. (More will be said later about GOSUBs 1ØØØ, 3ØØØ, and 6,ØØØ.)

```
81ØØ IF A$ = "1" OR A$ = "2" OR A$
     = "3" OR A$ = "4" THEN
     GOTO 814Ø

811Ø PRINT AT 15,3; A$; AT 15,6; "IS
     AN INVALID SELECTION"; AT
     16,4; "PRESS ENTER AND TRY
     AGAIN"

812Ø PAUSE 32767

813Ø GOTO 8ØØØ

814Ø IF A$ = "1" THEN GOSUB 1ØØØ

815Ø IF A$ = "2" THEN GOSUB 3ØØØ

816Ø IF A$ = "3" THEN GOSUB 6ØØØ

817Ø IF A$ = "4" THEN RETURN

818Ø GOTO 8ØØØ
```

Note that in this logic, we first test to see if a valid number was entered. If not, a message is displayed which shows the user this invalid entry and tells them to try again. The PAUSE statement gives them time to read the message before branching back to redisplay the menu.

If a valid response was entered, the program branches via GOSUBs to the applicable subsection of our program. A response of "4" (to end the program) returns control back to the caller of the "display menu" routine (line 1Ø) and the program will then be terminated by the STOP instruction on line 2Ø. Note that, after branching to the various subroutines, we GOTO the section which prints the menu, again. After all, the user may wish to continue the execution of the program by selecting another function.

Before we enter this menu-driven routine, what will happen when a response of "1", "2", or "3" is entered? Although the program calls for subroutines 1ØØØ, 3ØØØ and 6ØØØ to be branched to, we have not yet coded these

subsections. It is the intent of this Study Unit to explain how these routines are to be designed. But before we do this, it would be nice to run the menu part of the program to be sure that the proper branching occurs. A technique known as "stub testing" will allow us to do just this!

STUB TESTING

As programs become more and more complex, it helps to be able to test portions of the program at a time. That way, we can debug one section at a time without having to enter the entire program first.

A "stub" is a subroutine which can be branched to via a GOSUB and does little more than return to the caller.

For example, if we were to enter lines 1000, 3000 and 6000 as RETURN statements, our program could run smoothly, even though nothing would really happen.

A better way to code the stub modules would be to display a message on the screen which would let us know that the proper branching occurred before returning to the caller. See how we could code the subroutine for a response of "1" (ENTER APPOINTMENTS FOR A MONTH):

```
1000 REM ENTER ROUTINE STUB
1010 CLS
1020 PRINT "THIS IS THE ENTER
     ROUTINE"
1030 PAUSE 32767
1040 RETURN
```

Note that this stub just provides a *subroutine* to go to, *clears* the screen, *prints* a message so we can see that the proper branch occurred, *pauses* and, finally, *returns*. We can enter similar stubs for the UPDATE routine and the DISPLAY routine. This will give us the opportunity to RUN the program and debug it.

Later on, we can "fill out" each stub module, one at a time, when we are ready to do so. After we have coded the complete module, we can test it. By the time we have coded the entire program, the only bugs left will be those which relate to the interaction between two different subsections, thus making the entire application considerably easier to develop.

Let's now code and run this menu-driving portion as follows:

```
10 GOSUB 8000
20 STOP
1000 REM ENTER ROUTINE STUB
1010 CLS
1020 PRINT "THIS IS THE ENTER
     ROUTINE"
1030 PAUSE 32767
1040 RETURN
3000 REM UPDATE ROUTINE STUB
3010 CLS
3020 PRINT "THIS IS THE UPDATE
     ROUTINE"
3030 PAUSE 32767
3040 RETURN
6000 REM DISPLAY ROUTINE STUB
6010 CLS
6020 PRINT "THIS IS THE DISPLAY
     ROUTINE"
6030 PAUSE 32767
6040 RETURN
8000 REM DISPLAY MENU
```

```
8Ø1Ø CLS

8Ø2Ø PRINT AT Ø,4; "APPOINTMENTS
     CALENDAR"

8Ø3Ø PRINT AT 2,Ø; "NUMBER"; AT
     2,16; "FUNCTION"

8Ø4Ø PRINT AT 4,3; "1"; AT 4,12;
     "ENTER APPOINTMENTS"; AT
     5,13; "FOR A MONTH"

8Ø5Ø PRINT AT 7,3; "2"; AT 7,12;
     "CHANGE OR REMOVE"; AT
     8,13; "APPOINTMENTS"

8Ø6Ø PRINT AT 1Ø,3; "3"; AT 1Ø,12;
     "DISPLAY APPOINTMENTS"; AT
     11,13; "BY DAY OR NAME"

8Ø7Ø PRINT AT 13,3; "4"; AT 13,12;
     "END THE PROGRAM"

8Ø8Ø PRINT AT 21,Ø; "ENTER THE
     NUMBER OF YOUR CHOICE"

8Ø9Ø INPUT A$

81ØØ IF A$ = "1" OR A$ = "2" OR A$
     = "3" OR A$ = "4" THEN
     GOTO 814Ø

811Ø PRINT AT 15,3; A$; AT 15,6; "IS
     AN INVALID SELECTION"; AT
     16,4; "PRESS ENTER AND TRY
     AGAIN"

812Ø PAUSE 32767

813Ø GOTO 8ØØØ

814Ø IF A$ = "1" THEN GOSUB 1ØØØ

815Ø IF A$ = "2" THEN GOSUB 3ØØØ

816Ø IF A$ = "3" THEN GOSUB 6ØØØ

817Ø IF A$ = "4" THEN RETURN

818Ø GOTO 8ØØØ
```

CODING THE APPOINTMENTS ENTRY SUBROUTINE

Let us now tackle the stub module, which is executed when option "1" is chosen from an appointments calendar menu. This subroutine will dimension and initialize tables for each of the data items we wish our calendar to maintain.

TABLES

DAY	TIME	AM/PM	PLACE	NAME	NOTES
Ø1	9:ØØ	AM	86 5TH AVE.	JOE BROWN	JOB INTERVIEW
Ø1	12:ØØ	NOON	BILLS DINER	MARY SMITH	LUNCH
Ø2	8:3Ø	PM	11 N. ELM ST.	BILL JONES	PARTY
Ø6	11:3Ø	AM	3626 CENTRAL	DR. JOHNSON	CHECK-UP

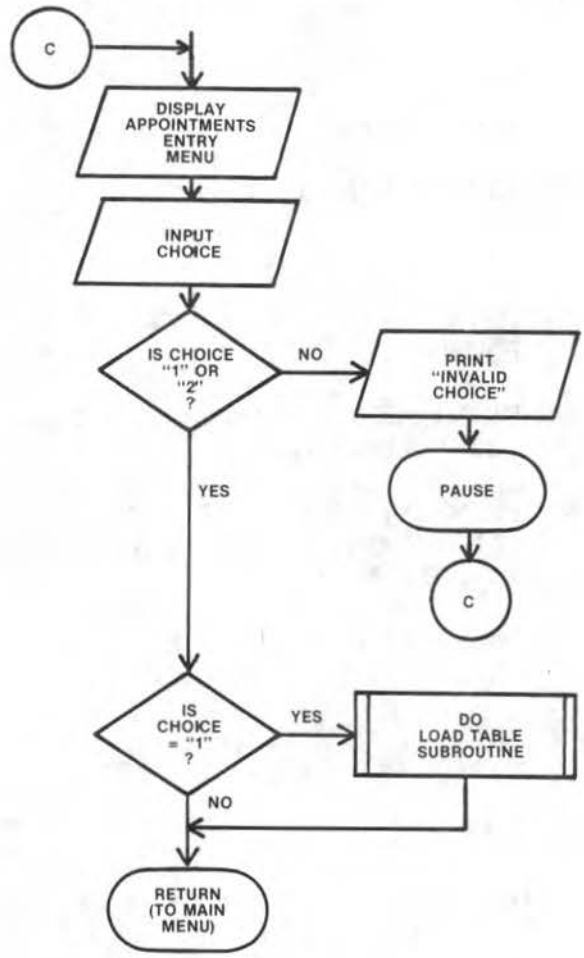
After we set up our table sizes, we'll have a FOR...NEXT loop to enter values for each of our appointments. This coding should be rather familiar to us by now. This section will be entered only once for each new month. After completing this module, the user will be reminded to set up a blank cassette tape so that the data can be saved. After the save instruction, the next step will be to display the menu, ensuring that the table data won't be lost when the tape is reloaded (i.e., a RUN command need not be entered; the program will run itself).

Here is how this module will be coded:

```
1000 REM ENTER ROUTINE
1010 CLS
1020 PRINT AT 0,5; "APPOINTMENTS
     ENTRY MENU"
1030 PRINT AT 3,0; "NUMBER"; AT
     3,12; "FUNCTION"
1040 PRINT AT 5,3; "1"; AT 5,12;
     "ENTER DATA FOR A NEW"; AT
     6,13; "MONTH"
1050 PRINT AT 8,3; "2"; AT 8,12;
     "RETURN TO MAIN MENU"
1060 PRINT AT 21,0; "ENTER
     NUMBER OF YOUR CHOICE"
1070 INPUT B$
1080 IF NOT B$ = "1" AND NOT B$ =
     "2" THEN GOTO 1110
1090 IF B$ = "1" THEN GOSUB 1200
1100 RETURN
1110 PRINT AT 20,0; B$; " IS NOT
     VALID—PRESS ENTER"
1120 PAUSE 32767
1130 GOTO 1010
1200 REM INITIALIZE TABLES
1210 DIM D(31)
1220 DIM T$(31,8)
1230 DIM P$(31,20)
1240 DIM N$(31,20)
1250 DIM M$(31,20)
1260 LET N = 0
1270 FOR L = 1 TO 31
1280 CLS
1290 PRINT AT 1,0; "ENTER DAY
     (01-31)"
1300 INPUT DAY
1310 IF DAY > 0 AND DAY < 32
     THEN GOTO 1360
1320 PRINT AT 1,0; "INVALID DAY—
     PRESS ENTER"
1330 PAUSE 32767
1340 GOTO 1280
1360 PRINT AT 1,19; DAY
1370 PRINT AT 3,0; "ENTER TIME"
1380 INPUT C$
1390 PRINT AT 3,12; C$
1400 PRINT AT 5,0; "ENTER PLACE"
1410 INPUT D$
1420 PRINT AT 5,13; D$
1430 PRINT AT 7,0; "ENTER NAME"
1440 INPUT E$
1450 PRINT AT 7,12; E$
```

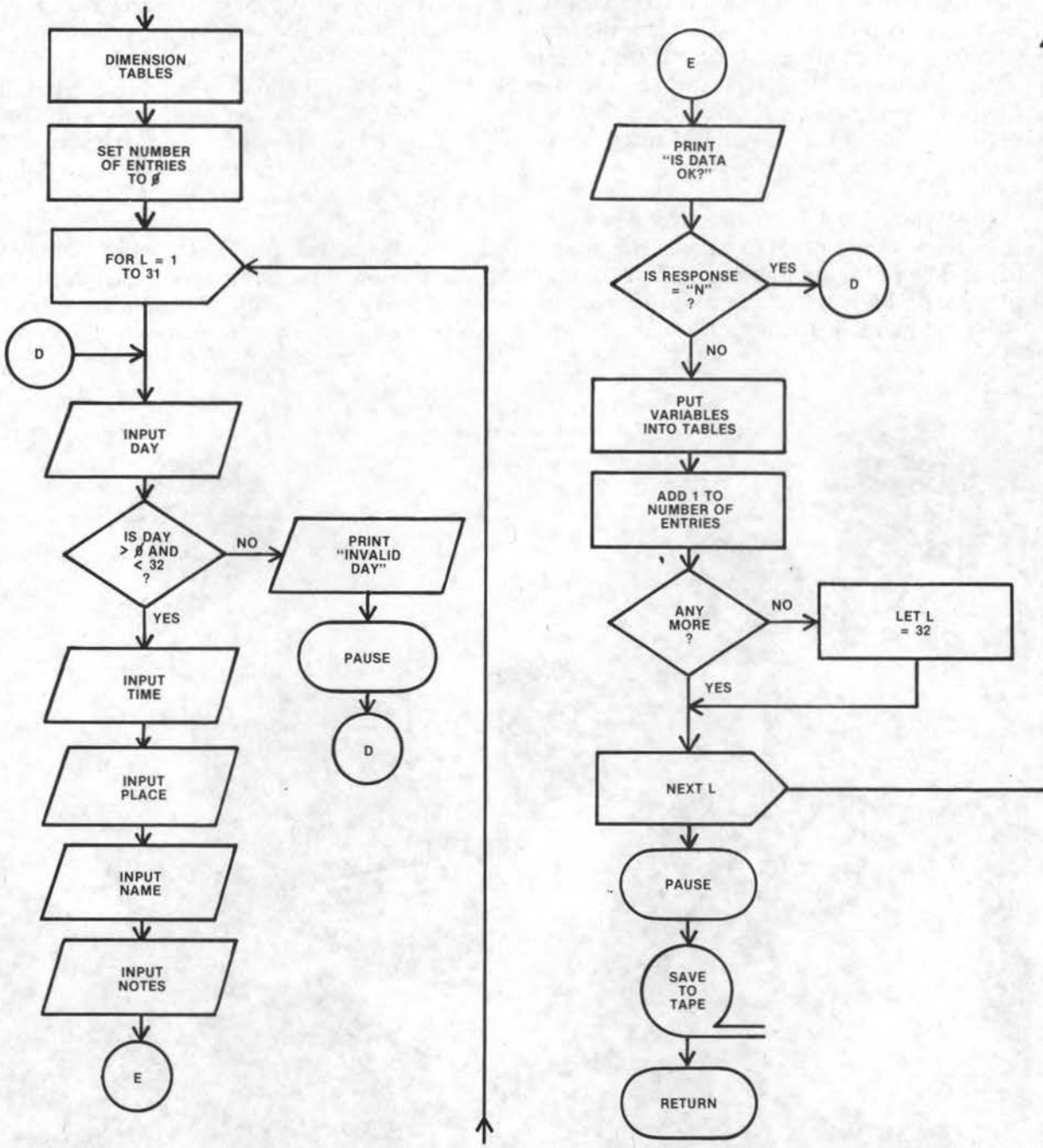
```
146Ø PRINT AT 9,Ø; "ENTER NOTES"
147Ø INPUT F$
148Ø PRINT AT 9,13; F$
149Ø PRINT AT 11,Ø; "PRESS ENTER
     IF OK—ELSE ENTER N"
15ØØ INPUT G$
151Ø IF G$ = "N" THEN GOTO 128Ø
152Ø LET D(L) = DAY
153Ø LET T$(L) = C$
154Ø LET P$(L) = D$
155Ø LET N$(L) = E$
156Ø LET M$(L) = F$
157Ø LET N = N + 1
158Ø PRINT AT 21,Ø; "ANY MORE
     (Y/N)?"
159Ø INPUT H$
16ØØ IF H$ = "N" THEN LET L = 32
161Ø NEXT L
162Ø CLS
163Ø PRINT AT 3,Ø; "SET TAPE TO
     RECORD—PRESS ENTER"
164Ø PAUSE 32767
165Ø SAVE "CALENDAR"
167Ø RETURN
```

ENTER ROUTINE



The flowchart design that was used to code this subroutine is as follows:

LOAD TABLE SUBROUTINE



When option "1" is selected from the main menu, the Enter Routine module is executed. This module displays its own menu—the Appointments Entry Menu. Here the user is given two choices: "1" will allow the user to enter a new month's records; "2" will return to the main menu. It is important that we give the user the second option, for as soon as option "1" is taken, any data stored in the tables would be lost.

An entry of "1" initializes and dimensions five tables, each with 31 locations. D is a table of the days of the appointments; T$ is the time of the day; P$ is the location of the meeting; N$ is the name of the person to be met; and M$ is the table of any notes the user might wish to keep regarding the meeting. Note that each of the character data tables has been given 2Ø bytes for each element (except for the time table, which has only eight).

Next, a counter is set to "Ø" (N). This counter will be added to for each record placed in the table. The other modules will utilize the value of N in order to know how big this month's table actually is.

We now enter a FOR. . .NEXT loop which will allow for a maximum of 31 records to be entered for a month. Prompts are issued for each variable. When the day of the month is

FIGURE 6—A computer and programs are tools to perform a job. Responsibility for quality of input and output is still in the hands of people such as programmers, decision makers, and operators.

entered, it is checked for validity—that is, a day between Ø1 and 31, inclusive. If a day outside of this range is submitted, the user will be informed of the error and given another chance to enter a valid day.

Notice that each value has not been immediately put into the tables; rather, they have been put into DAY, C$, D$, E$ and F$. After all values have been given, the user is issued a message to press ENTER if all the data is good; otherwise, the user will have to enter the record over again.

Here is another example of user-friendliness. Whenever possible, the user can either correct mistakes or change the data without having to go back to the beginning or, even worse, causing the program to blow up.

Each value is then positioned in the tables according to the value of the control variable and subscript, "L". At this point, our total number of records entered is incremented by one and a prompt is displayed for any more entries. If there are more, statement 161Ø (NEXT L) will add "1" to the control variable and branch back to the FOR L = 1 TO 31 instruction. Otherwise, "L" will be set to "32", thus terminating the FOR...NEXT loop.

When all appointments have been entered, the screen will be cleared, and the user "told" to set up a blank cassette tape to store the results.

Then line 167Ø will return to the caller (line 1Ø9Ø), where the next statement will cause a return to the original caller and produce the main menu.

As soon as you understand the logic of this module, enter this coding as a replacement for the stub we previously coded for the table entry subsection. Then, run the program and enter in the ten test data records that follow. Be sure to prepare your tape cassette to record this program; you will need it later, after we fill out our two other stubs.

Now pause for awhile and complete Programmer's Check #1 which follows. Verify your program before proceeding.

TEST DATA

DAY	TIME	PLACE	NAME	NOTES
Ø8	9:ØØ AM	1Ø1 1ST AVE.	ABC CO.	JOB INTERVIEW
Ø6	12 NOON	BILLS DINER	JOANNE	LUNCH
26	6:ØØ PM	15 CENTRAL	SAM AND DIANE	DINNER PARTY
15	1Ø:ØØ AM	1686 ELM ST.	XYZ BANK	CAR LOAN
Ø1	4:3Ø PM	AIRPORT	ALLIED AIRLINES	PICK UP MARY
Ø6	6:3Ø PM	PARIS CAFE	JOANNE	DINNER
21	7:3Ø PM	HOME	MIKE SMITH	CONSULTATION
15	1:3Ø PM	151 N. OAK ST.	DR. JOHNSON	CHECK-UP
Ø6	3:ØØ PM	CITY PARK	JOHNNY	LITTLE LEAGUE GAME
31	8:ØØ AM	816 MAPLE AVE.	AAA CAR SERVICE	TUNE-UP

PROGRAMMER'S CHECK

1

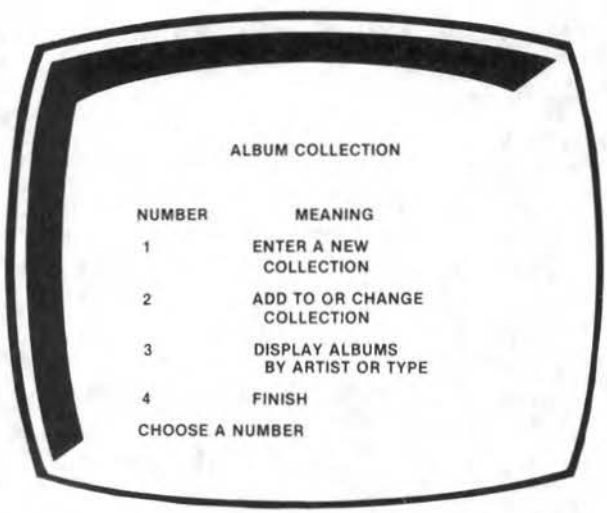
Using Stub Modules

Program Name: **IU1ØA1 (Instruction Unit 1Ø, Assignment 1)**

Type: **MENU-DRIVEN RECORD LIBRARY**

Specifications:

Design, code and debug a program which will be used to maintain files of music albums. The software should be menu-driven. The main menu should appear as follows:



Options 3 and 4 are to be coded as stub modules. The data to be maintained and entered is:

TABLES

ALBUM TITLE	ARTIST	CATALOG NO.	HIT SONGS	TYPE*

***Types of albums should use the following codes:**

RB — RHYTHM AND BLUES

JZ — JAZZ

RR — ROCK AND ROLL

CL — CLASSICAL

PP — POPULAR

GS — GOSPEL

CO — COMEDY

CW — COUNTRY AND WESTERN

MS — MISCELLANEOUS

Set up enough space for a 5Ø-album collection. Enter in about 3Ø records. Remember to SAVE it onto tape; we'll fill out the stubs in a later assignment.

(Answers on Pages 16-18)

MAINTAINING TABLE FILES

Now we should be ready to begin designing our next stub module—option "2" from our main menu, which is supposed to change or remove data in the tables.

There are actually three general categories of transactions which can be made to an existing table file:

- *Additions* to the file.
- *Deletions* from the file.
- *Changes* to records on the file.

ADDITIONS TO THE FILE

As long as our table is not full, we should allow the user to add new appointments made after the original tables were entered. We'll have to ensure, however, that the new entries are made into unoccupied positions of the tables. This is known as "merging".

Merging is the process of putting two files together. In our sample program, we will be merging records into the file one at a time. More often, in larger data processing applications, the additions to the file will be collected in a separate file, first. Then, the two files will be merged.

DELETIONS FROM THE FILE

In order to delete a record from the file, we must first be able to locate it in the table. We'll have to code logic which will search for the entry we wish to remove. Then, we can set the numeric variables to "ø" and the character variables to *blanks*. This process is actually a logical deletion—that is, there will still be a space where the record used to be, but our program logic will consider any blank positions to be "non-existent".

"Purging" is a process used to physically remove records from a file and to push all records which follow the deleted record up to fill in the blanks. Note the difference in the following illustration:

ORIGINAL FILE		LOGICAL DELETION		PHYSICAL DELETION	
NAME	PHONE #	NAME	PHONE #	NAME	PHONE #
JOE	991-1212	JOE	991-1212	JOE	991-1212
MARY	8ø6-9988	~~bbb~~	~~øøø-øøøø~~	SAM	772-2311
SAM	772-2311	SAM	772-2311	BETH	555-ø618
BETH	555-ø618	BETH	555-ø618		

PROGRAMMER'S CHECK ANSWERS

1

```
  10 GOSUB 8000
  20 STOP
1000 REM ENTER ROUTINE
1010 CLS
1020 PRINT AT 0,9;"ALBUM ENTRY M
ENU"
1030 PRINT AT 3,0;"NUMBER";AT 3,
12;"MEANING"
1040 PRINT AT 5,3;"1";AT 5,12;"E
NTER DATA FOR A NEW";AT 6,13;"CO
LLECTION"
1050 PRINT AT 8,3;"2";AT 8,12;"R
ETURN TO MAIN MENU"
1060 PRINT AT 21,0;"ENTER NUMBER
 OF YOUR CHOICE"
1070 INPUT B$
1080 IF NOT B$="1" AND NOT B$="2
" THEN GOTO 1110
1090 IF B$="1" THEN GOSUB 1200
1100 RETURN
1110 PRINT AT 20,0;B$;" IS NOT V
ALID-PRESS ENTER"
1120 PAUSE 32767
1130 GOTO 1010
1200 REM INITIALIZE TABLES
1210 DIM T$(50,20)
1220 DIM A$(50,20)
1230 DIM P$(50,10)
1240 DIM S$(50,20)
1250 DIM M$(50,2)
1260 LET N=0
1270 FOR L=1 TO 50
1280 CLS
1290 PRINT AT 1,0;"ENTER TITLE"
1300 INPUT U$
1360 PRINT AT 1,17;U$
1370 PRINT AT 3,0;"ENTER ARTIST"
1380 INPUT C$
1390 PRINT AT 3,17;C$
1400 PRINT AT 5,0;"ENTER CAT.NO.
"
1410 INPUT D$
1420 PRINT AT 5,17;D$
1430 PRINT AT 7,0;"ENTER HIT SON
G"
1440 INPUT E$
1445 PRINT AT 7,17;E$
1450 PRINT AT 9,0;"ENTER CODE TY
PE"
1470 INPUT F$
1480 PRINT AT 9,17;F$
1490 PRINT AT 11,0;"PRESS ENTER
IF OK-ELSE ENTER N"
1500 INPUT G$
1510 IF G$="N" THEN GOTO 1280
1520 LET T$(L)=U$
1530 LET A$(L)=C$
1540 LET P$(L)=D$
1550 LET S$(L)=E$
1560 LET M$(L)=F$
1570 LET N=N+1
1580 PRINT AT 21,0;"ANY MORE?(Y/
N)"
1590 INPUT H$
1600 IF H$="N" THEN LET L=50
1610 NEXT L
1620 CLS
1630 PRINT AT 3,0;"SET TAPE TO R
ECORD-PRESS ENTER"
1640 PAUSE 32767
1650 SAVE "ALBUMS"
1670 RETURN
3000 REM MODIFICATIONS ROUTINE
3010 CLS
3020 PRINT AT 0,3;"COLLECTION CH
ANGES"
3030 PRINT AT 2,0;"NUMBER";AT 2,
16;"MEANING"
3040 PRINT AT 4,3;"1";AT 4,12;"A
DD NEW";AT 5,12;"ALBUMS"
3050 PRINT AT 7,3;"2";AT 7,12;"R
EMOVE OLD";AT 8,12;"ALBUMS"
3060 PRINT AT 10,3;"3";AT 10,12;
"CHANGE OLD";AT 11,12;"ALBUMS"
3070 PRINT AT 13,3;"4";AT 13,12;
"RETURN TO MAIN MENU"
3080 PRINT AT 21,0;"ENTER THE NU
MBER OF YOUR CHOICE"
3090 INPUT B$
3100 IF B$="1" OR B$="2" OR B$="
3" OR B$="4" THEN GOTO 3140
3110 PRINT AT 15,3;B$;" IS AN IN
VALID SELECTION";AT 16,4;"PRESS
ENTER AND TRY AGAIN"
3120 PAUSE 32767
3130 GOTO 3010
3140 IF B$="1" THEN GOSUB 3200
3150 IF B$="2" THEN GOSUB 4000
3160 IF B$="3" THEN GOSUB 4500
3170 IF B$="4" THEN RETURN
3180 GOTO 3010
3200 REM ADDITIONS SUBROUTINE
3210 CLS
3220 PRINT "THERE IS ROOM FOR";N
-50;" MORE ENTRIES"
3230 IF 50-N>0 THEN GOTO 3260
3240 PAUSE 32767
3250 GOTO 3580
3260 PRINT AT 1,0;"ENTER TITLE"
3270 INPUT U$
3320 PRINT AT 1,17;U$
3330 PRINT AT 3,0;"ENTER ARTIST"
3340 INPUT C$
3350 PRINT AT 3,17;C$
3360 PRINT AT 5,0;"ENTER CAT.NO.
"
3370 INPUT D$
3380 PRINT AT 5,17;D$
3390 PRINT AT 7,0;"ENTER HIT SON
G"
3400 INPUT E$
3410 PRINT AT 7,17;E$
```

Programmer's Check 1 Answer (continued)

```
3420 PRINT AT 9,0;"ENTER CODE TY
PE"
3430 INPUT F$
3440 PRINT AT 9,17;F$
3450 PRINT AT 11,0;"PRESS ENTER
IF OK-ELSE ENTER N"
3460 INPUT G$
3470 IF G$="N" THEN GOTO 3210
3480 LET N=N+1
3490 LET T$(N)=U$
3500 LET A$(N)=C$
3510 LET P$(N)=D$
3520 LET S$(N)=E$
3530 LET M$(N)=F$
3540 PRINT AT 21,0;"ANY MORE?(Y/
N)"
3550 INPUT H$
3560 IF H$="N" THEN GOTO 3580
3570 GOTO 3210
3580 CLS
3590 PRINT AT 3,0;"SET TAPE TO R
ECORD-PRESS ENTER"
3600 PAUSE 32767
3610 SAVE "ALBUM$"
3620 RETURN
4000 REM REMOVE OLD ALBUMS
4005 CLS
4010 PRINT AT 1,5;"REMOVE OLD AL
BUMS"
4015 PRINT AT 5,0;"ENTER TITLE T
O BE REMOVED"
4020 INPUT R$
4025 LET S=1
4030 IF R$=T$(S,1 TO LEN R$) THE
N GOTO 4060
4035 LET S=S+1
4040 IF S>N THEN GOTO 4050
4045 GOTO 4030
4050 PRINT AT 5,0;R$;" NOT ON LI
ST-PRESS ENTER"
4053 PAUSE 32767
4055 GOTO 4160
4060 CLS
4065 PRINT AT 2,3;"ALBUM TO BE R
EMOVED"
4070 PRINT AT 4,0;"TITLE: ";R$
4075 PRINT AT 6,0;"ARTIST:";A$(S
)
4080 PRINT AT 8,0;"CAT.NO.:";P$(
S)
4085 PRINT AT 10,0;"HIT SONG:";S
$(S)
4090 PRINT AT 12,0;"CODE TYPE:";
M$(S)
4095 PRINT AT 20,0;"ENTER Y TO R
EMOVE - ELSE N"
4100 INPUT X$
4110 IF X$<>"Y" THEN GOTO 4160
4120 LET T$(S)=""
4130 LET A$(S)=""
4140 LET P$(S)=""
4150 LET S$(S)=""
4155 LET M$(S)=""
4160 CLS
4165 PRINT AT 21,0;"ANY MORE?(Y/
N)"
4170 INPUT X$
4180 IF X$="Y" THEN GOTO 4000
4190 RETURN
4500 REM CHANGE MODULE
4505 CLS
4510 PRINT AT 1,6;"CHANGE MENU"
4515 PRINT AT 3,0;"NUMBER";AT 3,
10;"MEANING"
4520 PRINT AT 6,0;"1";AT 6,10;"C
HANGE TITLE"
4525 PRINT AT 7,0;"2";AT 7,10;"C
HANGE ARTIST"
4530 PRINT AT 8,0;"3";AT 8,10;"C
HANGE CAT.NO."
4535 PRINT AT 9,0;"4";AT 9,10;"C
HANGE HIT SONG"
4540 PRINT AT 10,0;"5";AT 10,10;
"CHANGE CODE TYPE"
4545 PRINT AT 12,0;"6";AT 12,10;
"RETURN TO MENU"
4550 PRINT AT 21,0;"CHOOSE A NUM
BER"
4555 INPUT R$
4560 IF R$="6" THEN RETURN
4565 IF R$<"0" OR R$>"5" THEN GO
TO 4500
4568 CLS
4570 PRINT AT 21,0;"ENTER OLD TI
TLE"
4575 INPUT Z$
4580 LET S=1
4585 IF Z$=T$(S,1 TO LEN Z$) THE
N GOTO 4625
4590 LET S=S+1
4600 IF S>N THEN GOTO 4610
4605 GOTO 4585
4610 CLS
4612 PRINT AT 5,0;Z$;" NOT ON FI
LE-PRESS ENTER"
4615 PAUSE 32767
4620 GOTO 4800
4625 CLS
4630 PRINT AT 3,0;"ALBUM TO BE C
HANGED:"
4635 PRINT AT 5,0;"TITLE:";T$(S)
4640 PRINT AT 6,0;"ARTIST:";A$(S
)
4645 PRINT AT 7,0;"CAT.NO.:";P$(
S)
4670 PRINT AT 8,0;"HIT SONG:";S$
(S)
4675 PRINT AT 9,0;"CODE TYPE:";M
$(S)
4680 PRINT AT 21,0;"ENTER CHANGE
"
4685 INPUT K$
```

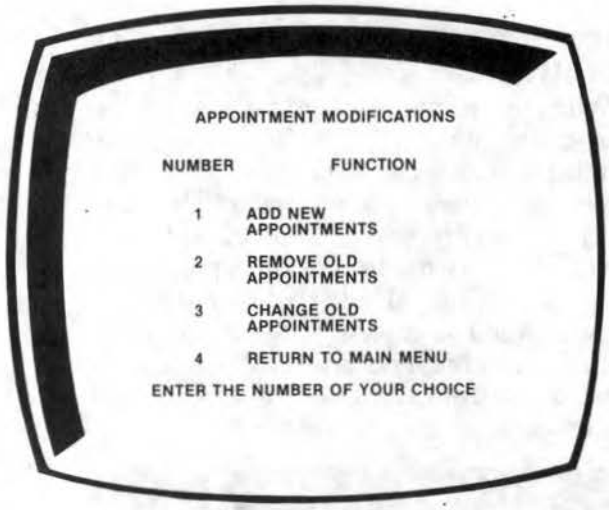
Programmer's Check 1 Answer (continued)

```
4690 IF R$="1" THEN LET T$(S)=K$
4700 IF R$="2" THEN LET A$(S)=K$
4710 IF R$="3" THEN LET P$(S)=K$
4720 IF R$="4" THEN LET S$(S)=K$
4730 IF R$="5" THEN LET M$(S)=K$
4800 GOTO 4500
5000 REM DISPLAY MODULE
5010 CLS
5020 PRINT AT 1,10;"DISPLAY ALBU
MS"
5030 PRINT AT 3,0;"CODE";AT 3,10
;"MEANING"
5040 PRINT AT 5,1;"A";AT 5,10;"D
ISPLAY ARTIST"
5050 PRINT AT 7,1;"C";AT 7,10;"D
ISPLAY BY CODE"
5060 PRINT AT 9,1;"R";AT 9,10;"R
ETURN TO MENU"
5070 PRINT AT 21,0;"ENTER CODE"
5100 INPUT R$
5110 IF R$="R" THEN RETURN
5120 IF R$="A" THEN GOTO 5150
5130 IF R$="C" THEN GOTO 5400
5140 GOTO 5000
5150 CLS
5160 PRINT AT 1,0;"ENTER ARTIST"
5170 INPUT Q$
5180 LET S=1
5190 IF Q$=A$(S,1 TO LEN Q$) THE
N GOTO 5230
5200 LET S=S+1
5210 IF S>N THEN GOTO 5310
5220 GOTO 5190
5230 CLS
5240 PRINT AT 1,0;Q$
5250 PRINT AT 3,0;"TITLE:";T$(S)
5260 PRINT AT 4,0;"CAT.NO.:";P$(
S)
5270 PRINT AT 5,0;"HIT SONG:";S$
(S)
5280 PRINT AT 6,0;"CODE TYPE:";M
$(S)
5290 PAUSE 32767
5300 GOTO 5200
5310 CLS
5320 PRINT AT 5,16;"THATS IT"
5325 PAUSE 150
5330 GOTO 5000
5400 CLS
5410 PRINT AT 1,0;"CHOOSE CATEGO
RY"
5420 PRINT AT 3,0;"CODE";AT 3,10
;"MEANING"
5430 PRINT AT 5,0;"RB";AT 5,10;"
RHYTHM AND BLUES"
5440 PRINT AT 6,0;"JZ";AT 6,10;"
JAZZ"
5450 PRINT AT 7,0;"RR";AT 7,10;"
ROCK AND ROLL"
5460 PRINT AT 8,0;"CL";AT 8,10;"
CLASSICAL"
5470 PRINT AT 9,0;"PP";AT 9,10;"
POPULAR"
5480 PRINT AT 10,0;"GS";AT 10,10
;"GOSPEL"
5490 PRINT AT 11,0;"CO";AT 11,10
;"COMEDY"
5500 PRINT AT 12,0;"CW";AT 12,10
;"COUNTRY WESTERN"
5510 PRINT AT 13,0;"MS";AT 13,10
;"MISCELLANEOUS"
5520 PRINT AT 21,0;"ENTER CODE"
5530 INPUT Q$
5540 LET S=1
5550 IF Q$=M$(S) THEN GOTO 5590
5560 LET S=S+1
5570 IF S>N THEN GOTO 5670
5580 GOTO 5550
5590 CLS
5600 PRINT AT 1,10;Q$
5610 PRINT AT 3,0;"TITLE:";T$(S)
5620 PRINT AT 5,0;"ARTIST:";A$(S
)
5630 PRINT AT 7,0;"CAT.NO.:";P$(
S)
5640 PRINT AT 9,0;"HIT SONG:";S$
(S)
5650 PAUSE 32767
5660 GOTO 5560
5670 CLS
5680 PRINT AT 5,16;"THATS IT"
5690 PAUSE 150
5700 GOTO 5000
8000 REM DISPLAY MENU
8010 CLS
8020 PRINT AT 0,3;"ALBUM COLLECT
ION"
8030 PRINT AT 2,0;"NUMBER";AT 2,
16;"MEANING"
8040 PRINT AT 4,3;"1";AT 4,12;"E
NTER A NEW";AT 5,13;"COLLECTION"
8050 PRINT AT 7,3;"2";AT 7,12;"A
DD TO OR CHANGE";AT 8,13;"COLLEC
TION"
8060 PRINT AT 10,3;"3";AT 10,12;
"DISPLAY ALBUMS";AT 11,13;"BY AR
TIST OR TYPE"
8070 PRINT AT 13,3;"4";AT 13,12;
"FINISH"
8080 PRINT AT 21,0;"CHOOSE A NUM
BER"
8090 INPUT V$
8100 IF V$="1" OR V$="2" OR V$="
3" OR V$="4" THEN GOTO 8140
8110 PRINT AT 15,3;V$;AT 15,6;"I
S AN INVALID SELECTION";AT 16,4;
"PRESS ENTER AND TRY AGAIN"
8120 PAUSE 32767
8130 GOTO 8000
8140 IF V$="1" THEN GOSUB 1000
8150 IF V$="2" THEN GOSUB 3000
8160 IF V$="3" THEN GOSUB 5000
8170 IF V$="4" THEN RETURN
8180 GOTO 8000
```

CHANGES TO THE FILE

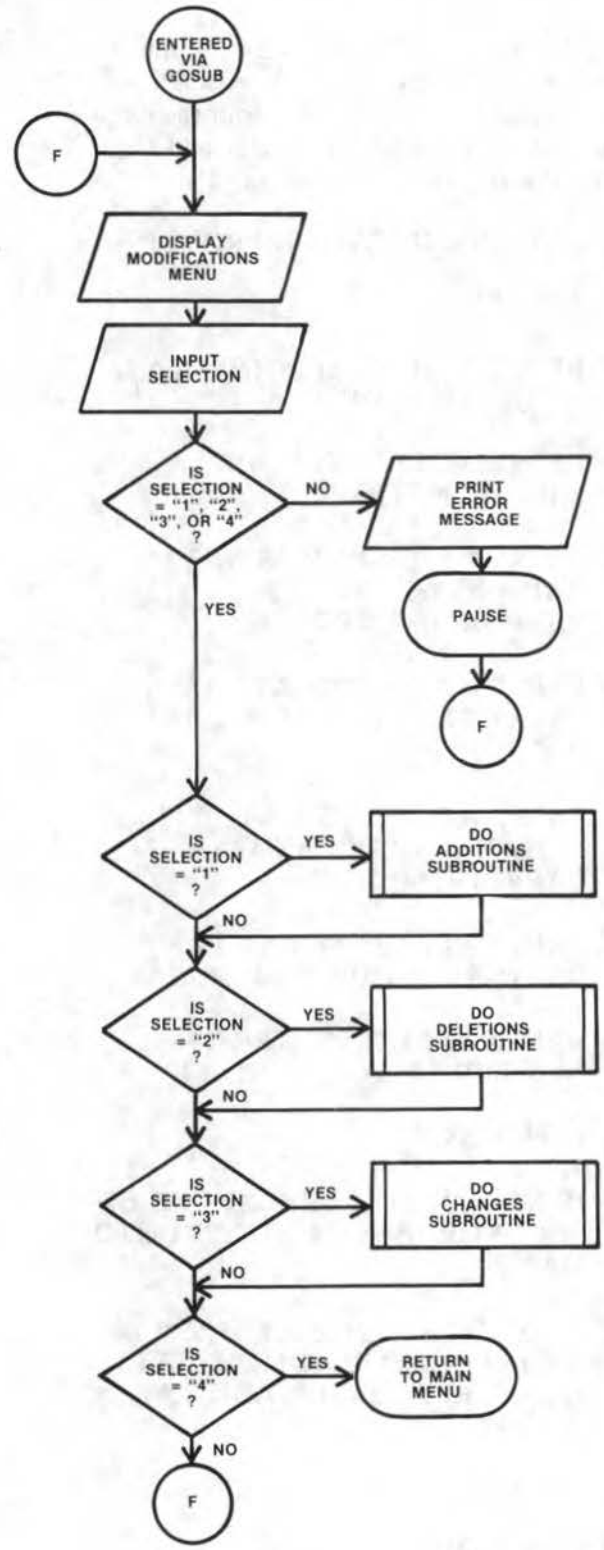
The third category of file maintenance involves changes to records on the file. We should allow the user to change any variable for any record desired. But, before the change can be made, the record must first be located on the file, as in making deletions.

As you can see, there are enough decisions that the user has to make when maintaining the file to warrant separate menus. So, let's design a menu which will be displayed when option "2" is entered from the appointments calendar menu.



After the PRINT statements, our logic will, once again, cause branching to the various sections of our modifications routine. This portion of the flowchart design is as follows:

MODIFICATIONS ROUTINE



This coding will appear very familiar to us by now. Indeed, entering it can be made simple by listing the line numbers from the main menu, changing the line numbers, and then, making the few changes necessary.

```
3000 REM MODIFICATIONS ROUTINE

3010 CLS

3020 PRINT AT 0,3; "APPOINTMENT
     MODIFICATIONS"

3030 PRINT AT 2,0; "NUMBER"; AT
     2,16; "FUNCTION"

3040 PRINT AT 4,3; "1"; AT 4,12;
     "ADD NEW"; AT 5,12;
     "APPOINTMENTS"

3050 PRINT AT 7,3; "2"; AT 7,12;
     "REMOVE OLD"; AT 8,12;
     "APPOINTMENTS"

3060 PRINT AT 10,3; "3"; AT 10,12;
     "CHANGE OLD"; AT 11,12;
     "APPOINTMENTS"

3070 PRINT AT 13,3; "4"; AT 13,12;
     "RETURN TO MAIN MENU"

3080 PRINT AT 21,0; "ENTER THE
     NUMBER OF YOUR CHOICE"

3090 INPUT B$

3100 IF B$ = "1" OR B$ = "2" OR B$
     = "3" OR B$ = "4" THEN GOTO
     3140

3110 PRINT AT 15,3; B$; AT 15,6; " IS
     AN INVALID SELECTION"; AT
     16,4; "PRESS ENTER AND TRY
     AGAIN"

3120 PAUSE 32767

3130 GOTO 3010

3140 IF B$ = "1" THEN GOSUB 3200

3150 IF B$ = "2" THEN GOSUB 4000

3160 IF B$ = "3" THEN GOSUB 4500

3170 IF B$ = "4" THEN RETURN

3180 GOTO 3010
```

Enter this coding to the master copy of your program (reload it from the tape, if necessary).

Once the menu is working, follow the logic and insert the lines into your program.

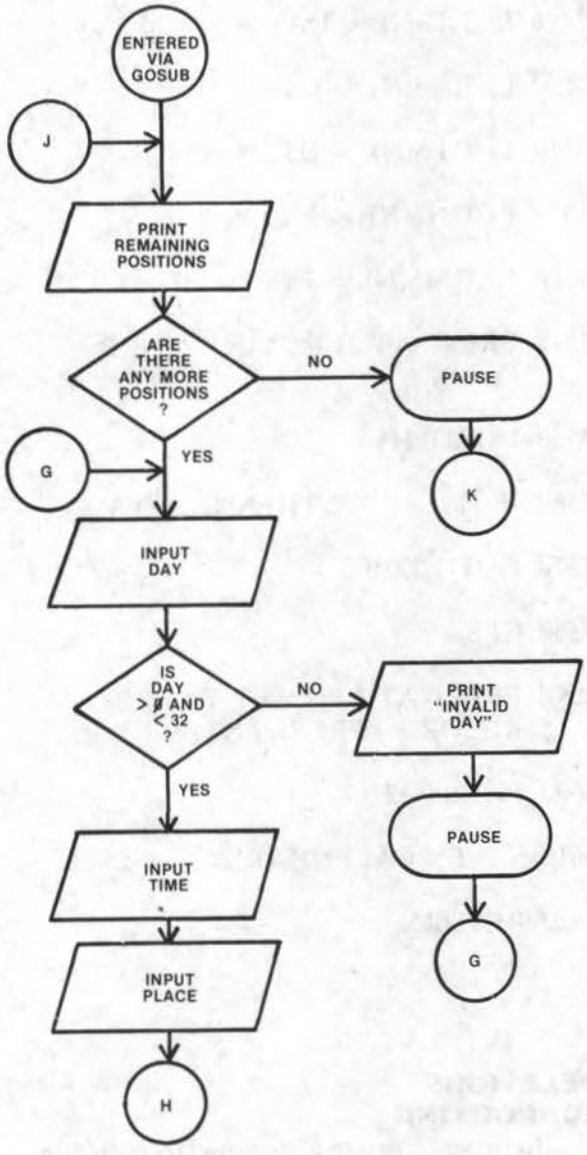
ADDITIONS SUBROUTINE

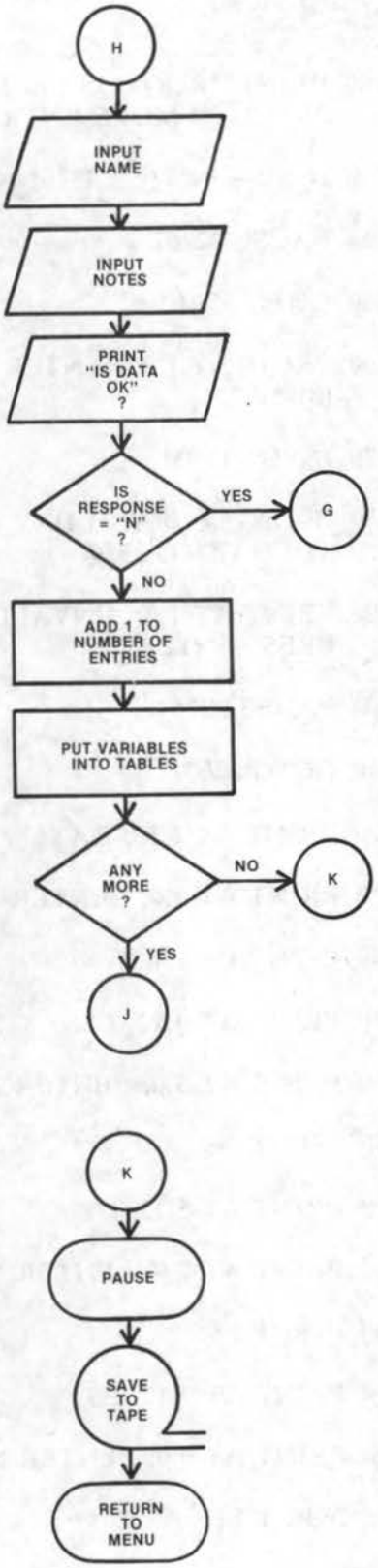
When the user needs to add new appointments to the master files, we'll first have to determine if there is any space remaining. Fortunately, our program has a variable. The total number of entries (N), which if subtracted from 31, will allow us to display the number of entries that can be added. If this number is "0", we should so inform the user and then branch back to the modifications menu. This will avoid confusing the user by not having the program come to an abnormal end.

FIGURE 7—Programming for business does require a familiarity with such things as office routines, sales reports, production schedules and inventory control. Most business managers are quite willing to explain their systems to the programmer. Often, the excecutives already have a plan for using a computer and software or simply need revisions to existing programs.

Next, our new design will pretty closely follow the logic required to enter the appointments previously, except we will *not* have to dimension the tables or set up a FOR...NEXT loop. We can just increment "N" by "1" and add the new value into the tables.

FLOWCHART DESIGN





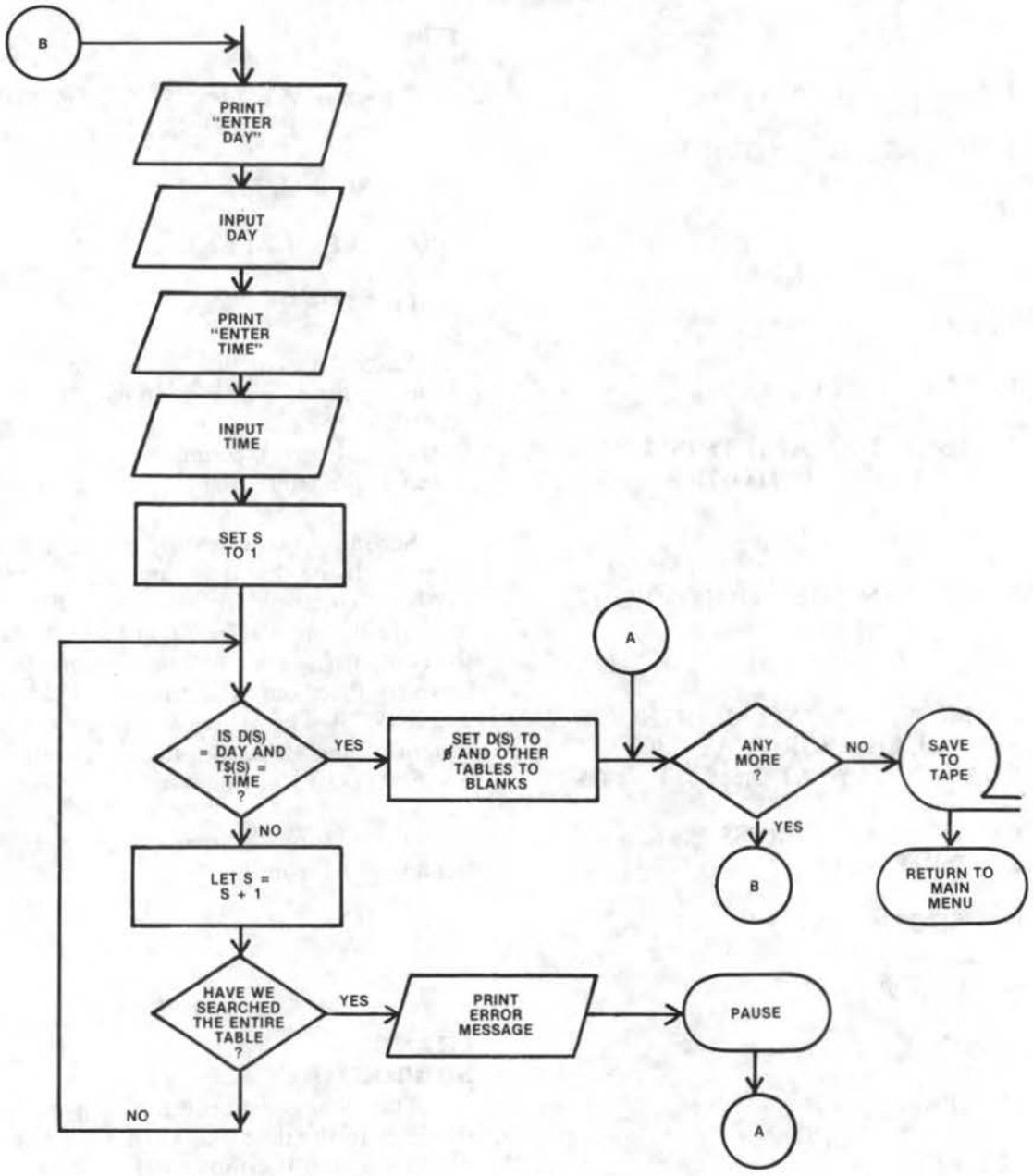
```
32ØØ REM ADDITIONS SUBROUTINE
321Ø CLS
322Ø PRINT "ROOM IS LEFT FOR ";
     31 - N; " MORE ENTRIES"
323Ø IF 31 - N > Ø THEN GOTO 326Ø
324Ø PAUSE 32767
325Ø GOTO 358Ø
326Ø PRINT AT 1,Ø; "ENTER DAY
     (Ø1-31)"
327Ø INPUT DAY
328Ø IF DAY > Ø AND DAY < 32
     THEN GOTO 332Ø
329Ø PRINT AT 1,Ø; "INVALID DAY -
     PRESS ENTER"
33ØØ PAUSE 32767
331Ø GOTO 326Ø
332Ø PRINT AT 1,19; DAY
333Ø PRINT AT 3,Ø; "ENTER TIME"
334Ø INPUT C$
335Ø PRINT AT 3,12; C$
336Ø PRINT AT 5,Ø; "ENTER PLACE"
337Ø INPUT D$
338Ø PRINT AT 5,13; D$
339Ø PRINT AT 7,Ø; "ENTER NAME"
34ØØ INPUT E$
341Ø PRINT AT 7,12; E$
342Ø PRINT AT 9,Ø; "ENTER NOTES"
343Ø INPUT F$
344Ø PRINT AT 9,13; F$
345Ø PRINT AT 11,Ø; "PRESS ENTER
     IF OK—ELSE ENTER N"
346Ø INPUT G$
347Ø IF G$ = "N" THEN GOTO 321Ø
348Ø LET N = N + 1
349Ø LET D(N) = DAY
35ØØ LET T$(N) = C$
351Ø LET P$(N) = D$
352Ø LET N$(N) = E$
353Ø LET M$(N) = F$
354Ø PRINT AT 21,Ø; "ANY MORE
     (Y/N)?"
355Ø INPUT H$
356Ø IF H$ = "N" THEN GOTO 358Ø
357Ø GOTO 321Ø
358Ø CLS
359Ø PRINT AT 3,Ø; "SET TAPE TO
     RECORD—PRESS ENTER"
36ØØ PAUSE 32767
361Ø SAVE "CALENDAR"
362Ø RETURN
```

DELETIONS SUBROUTINE

In order to delete a meeting from the file, we will prompt the user to enter the day and time of the meeting to be removed. Our logic will then search through the tables of days and times. When a match is found, the day will be

reset to zero and the time, place, name and notes reset to blanks. This will then be a logical deletion; the old record will still physically occupy a position in the file, but our program logic will treat it as if it were not there. The flowchart design for this subroutine follows:

DELETIONS SUBROUTINE



```
4ØØØ REM DELETIONS SUBROUTINE
4Ø1Ø CLS
4Ø2Ø PRINT AT 5,Ø; "ENTER DAY"
4Ø3Ø INPUT DAY
4Ø4Ø PRINT AT 5,11; DAY
4Ø5Ø PRINT AT 7,Ø; "ENTER TIME"
4Ø6Ø INPUT C$
4Ø7Ø PRINT AT 7,12; C$
4Ø8Ø LET S = 1
4Ø9Ø LET P = LEN C$
41ØØ IF D(S) = DAY AND T$ (S, 1 TO
     P) = C$ THEN GOTO 418Ø
411Ø LET S = S + 1
412Ø IF S < = N THEN GOTO 41ØØ
413Ø CLS
414Ø PRINT AT 1,Ø; "NO MEETING
     SCHEDULED FOR:"; AT 2,Ø;
     "DAY "; DAY; AT 2,8; "AT "; C$
415Ø PRINT AT 4,Ø; "PRESS ENTER
     TO CONTINUE"
416Ø PAUSE 32767
417Ø GOTO 423Ø
418Ø LET D(S) = Ø
419Ø LET T$(S) = " "
42ØØ LET P$(S) = " "
421Ø LET N$(S) = " "
422Ø LET M$(S) = " "
423Ø PRINT AT 21,Ø; "ANY MORE
     (Y/N)?"
424Ø INPUT H$
425Ø IF H$ = "N" THEN GOTO 427Ø
426Ø GOTO 4ØØØ
427Ø CLS
428Ø PRINT AT 3,Ø; "SET TAPE TO
     RECORD—PRESS ENTER"
429Ø PAUSE 32767
43ØØ SAVE "CALENDAR"
431Ø RETURN
```

Several instructions here are worth examining. First, while we have 31 potential records in the tables, we can limit our search to the number of meetings actually in the file, which is the value of N. (See line 412Ø.)

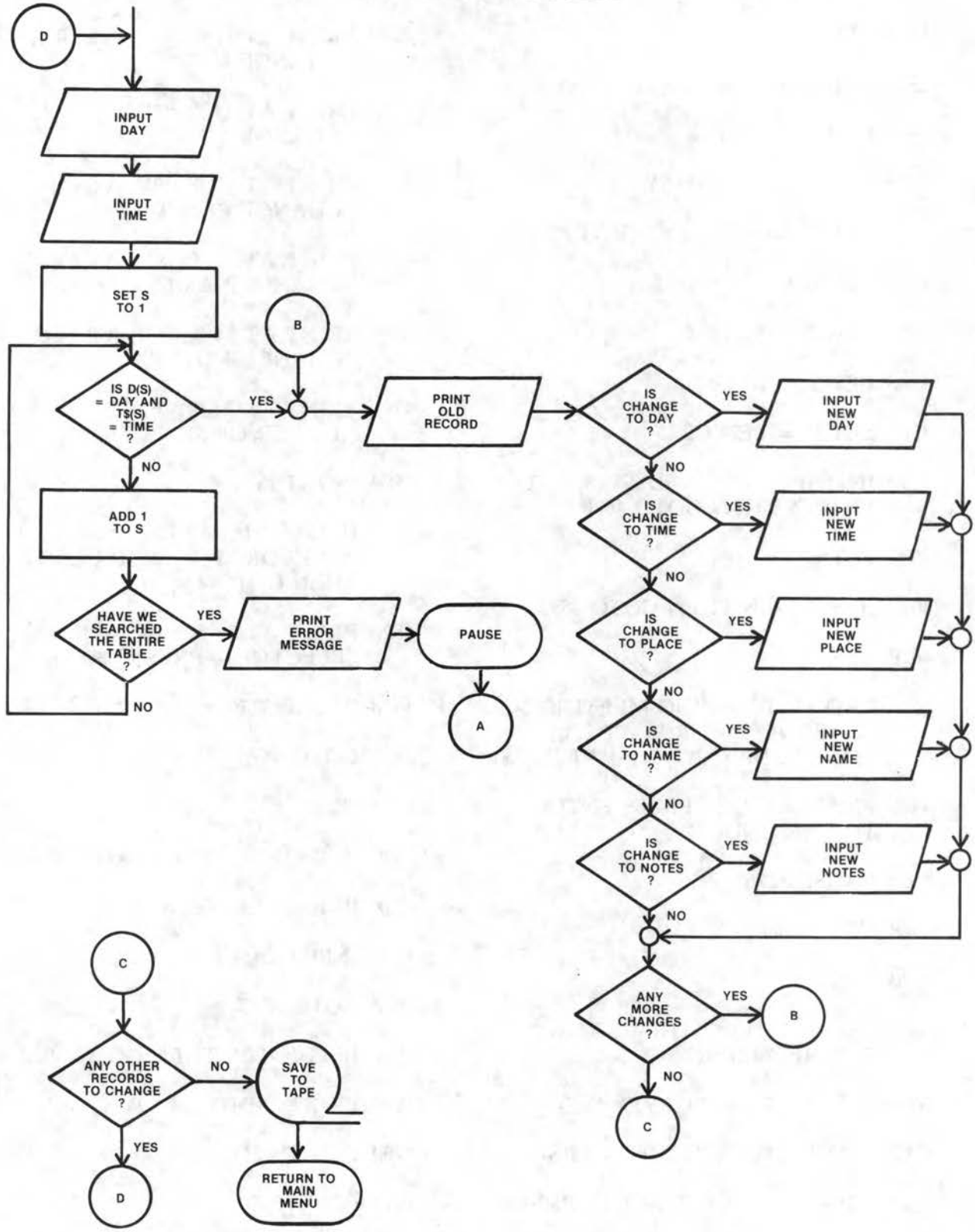
Secondly, we can only delete a meeting when both the day *and* time are found (line 41ØØ). Comparing for the day will be done numerically and will be no problem. However, the comparison for the time will only be valid if we compare only the number of characters entered (C$). Therefore, we can use the length function, line 4Ø9Ø, and then compare C$ to T$ (S, 1 TO P) in line 41ØØ.

We will now move on to coding the "change" subroutine.

CHANGE SUBROUTINE

The change subroutine will be coded similarly to the deletions in that we will again have to search the tables before making any modifications. The user will again be prompted for the day and time of the meeting to be changed. Once the entry is found in the file, we must then determine which fields the user must change. The users will then be allowed to change one or more fields in the tables. The flowchart for this module follows:

CHANGE SUBROUTINE



```
45ØØ REM CHANGES SUBROUTINE
451Ø CLS
452Ø PRINT AT 5,Ø; "ENTER DAY"
453Ø INPUT DAY
454Ø PRINT AT 5,11; DAY
455Ø PRINT AT 7,Ø; "ENTER TIME"
456Ø INPUT C$
457Ø PRINT AT 7,12; C$
458Ø LET S = 1
4585 LET P = LEN C$
459Ø IF D(S) = DAY AND T$ (S, 1 TO
     P) = C$ THEN GOTO 467Ø
46ØØ LET S = S + 1
461Ø IF S < = N THEN GOTO 459Ø
462Ø CLS
463Ø PRINT AT 1,Ø; "NO MEETING
     SCHEDULED FOR:"; AT 2,Ø;
     "DAY "; DAY; AT 2,8; "AT "; C$
464Ø PRINT AT 4,Ø; "PRESS ENTER
     TO CONTINUE"
465Ø PAUSE 32767
466Ø GOTO 5Ø5Ø
467Ø CLS
468Ø PRINT AT 1,Ø; "THIS IS THE
     OLD RECORD:"
469Ø PRINT AT 2,Ø; "DAY "; D(S)
47ØØ PRINT AT 3,Ø; "TIME "; T$(S)
471Ø PRINT AT 4,Ø; "PLACE "; P$(S)
472Ø PRINT AT 5,Ø; "NAME "; N$(S)
473Ø PRINT AT 6,Ø; "NOTES "; M$(S)
474Ø PRINT AT 1Ø,Ø; "1"; AT 1Ø,6;
     "CHANGE DAY"
475Ø PRINT AT 11,Ø; "2"; AT 11,6;
     "CHANGE TIME"
476Ø PRINT AT 12,Ø; "3"; AT 12,6;
     "CHANGE PLACE"
477Ø PRINT AT 13,Ø; "4"; AT 13,6;
     "CHANGE NAME"
478Ø PRINT AT 14,Ø; "5"; AT 14,6;
     "CHANGE NOTES"
479Ø PRINT AT 2Ø,Ø; "ENTER
     NUMBER OF SELECTION"
48ØØ INPUT I$
481Ø IF I$ = "1" OR I$ = "2" OR I$
     = "3" OR I$ = "4" OR I$ = "5"
     THEN GOTO 484Ø
482Ø PRINT AT 21,Ø; "INVALID
     SELECTION—PRESS ENTER"
483Ø PAUSE 32767
4835 GOTO 467Ø
4837 CLS
484Ø IF I$<>"1" THEN GOTO 488Ø
485Ø PRINT "ENTER NEW DAY"
486Ø INPUT D(S)
487Ø GOTO 5Ø2Ø
488Ø IF I$<>"2" THEN GOTO 492Ø
489Ø PRINT "ENTER NEW TIME"
49ØØ INPUT T$(S)
491Ø GOTO 5Ø2Ø
492Ø IF I$<>"3" THEN GOTO 496Ø
```

```
4930 PRINT "ENTER NEW PLACE"
4940 INPUT P$(S)
4950 GOTO 5020
4960 IF I$ < >"4" THEN GOTO 5000
4970 PRINT "ENTER NEW NAME"
4980 INPUT N$(S)
4990 GOTO 5020
5000 PRINT "ENTER NEW NOTES"
5010 INPUT M$(S)
5020 CLS
5025 PRINT AT 21,0; "ANY MORE
     CHANGES (Y/N)?"
5030 INPUT J$
5040 IF J$ = "Y" THEN GOTO 4670
5050 PRINT AT 21,0; "ANY OTHER
     RECORDS TO CHANGE?"
5060 INPUT H$
5070 IF H$ = "Y" THEN GOTO 4500
5080 CLS
5090 PRINT AT 3,0; "SET TAPE TO
     RECORD—PRESS ENTER"
5100 PAUSE 32767
5110 SAVE "CALENDAR"
5120 RETURN
```

Notice that in this logic, the record to be changed is first displayed as the user is requested to pick a number corresponding to the field that is to be altered. When a number is entered, the program branches to prompt the user for the new data. Then, the entry is placed over the old data in the table. The user is given an opportunity to change more fields on that record, if desired.

ENTER this addition to the program. RUN the entire program utilizing all of the options we have entered. SAVE the additions onto tape. This program has gotten quite lengthy—we wouldn't want to have to enter it all over again!

Familiarize yourself with the program as it now exists. Remember, only a short while ago we had merely coded stubs for the additions and modifications modules. Consider how difficult it would be to design and code this program without having developed it piece by piece as we have.

In the next section, we will complete the third and last major module of our appointments calendar program: the listing or displaying of meetings by day or by name. In order to produce this output most efficiently, it is best to first sort these records, since they may have been entered and updated randomly (in no sequence).

SORTING

Sorting is the process of arranging records into sequence. This sequence may be ascending (from lowest to highest) or descending (from highest to lowest) according to a particular key or field.

The method of sorting we will use is called a "bubble" sort because, like bubbles, the smallest (or lightest) values will float to the top of the table.

Let's first examine a bubble sort which will rearrange a table of six numeric values.

TABLE OF VALUES (UNSORTED)	TABLE OF VALUES (SORTED)
16	5
85	16
32	24
5	32
78	78
24	85

In this bubble sort, we will pass through the list of values five times (one less than the number of values to be sorted). On each pass, each element will be compared to the next. If the first is less than the second, then the two values are left alone, as they are already in ascending sequence. If, however, the first is greater than the second, then they will be switched: the first being put into the second position and vice versa.

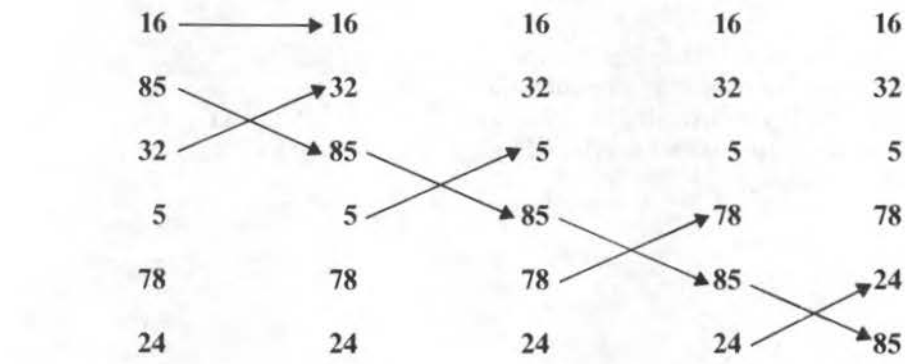
In this manner, the lower values will move towards the top of the table until all the values are in ascending sequence. Look at the following illustration to see how the table will appear after each pass.

Notice that the table in Figure 9 is sorted after only three passes. The fourth and fifth passes will not necessitate any switching. Depending on the initial values of the table, however, we may need all five passes to ensure that all values are sorted.

Now stop and complete Programmer's Check #2. Be sure you understand the concept of bubble sorting before continuing.

FIGURE 8—Executives can speak confidently when their reports are based upon sound data. The programs used to process data in a computer must be designed for accuracy and flexibility. The requirements of today will possibly change tomorrow. New modules should be anticipated so they can be added when necessary.

BUBBLE SORT PROCESSING



NOTE: After the first pass, the largest value, 85, is on the bottom of the table.

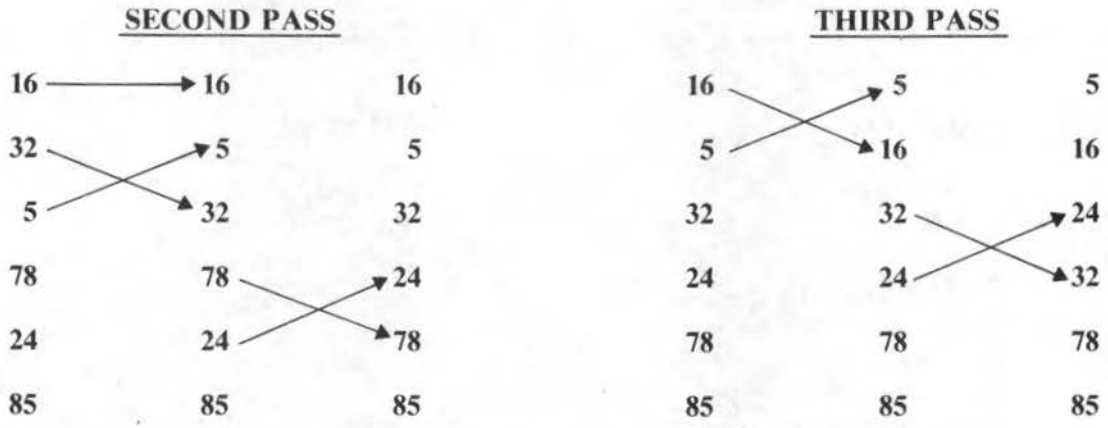


FIGURE 9—It takes three "passes" for these six random numeric values to be sorted and arranged in "bubble" format. However, it can take as many as five passes to accomplish this task, depending upon the arrangement.

PROGRAMMER'S CHECK

2

The Logic of "Bubble" Sorting

1. Try your hand at using the bubble sort process with the following numbers. Will you use all five passes to complete the task of sorting six numeric values?

FIRST PASS:

INITIAL VALUE					
9Ø	___	___	___	___	___
68	___	___	___	___	___
5Ø	___	___	___	___	___
29	___	___	___	___	___
18	___	___	___	___	___
7	___	___	___	___	___

SECOND PASS:

INITIAL VALUE				
___	___	___	___	___
___	___	___	___	___
___	___	___	___	___
___	___	___	___	___
___	___	___	___	___
___	___	___	___	___

THIRD PASS:

INITIAL VALUE			
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___

FOURTH PASS:

INITIAL VALUE		
___	___	___
___	___	___
___	___	___
___	___	___
___	___	___
___	___	___

FIFTH PASS:

INITIAL VALUE	
___	___
___	___
___	___
___	___
___	___
___	___

Programmer's Check 2 (continued)

2. **Want to try another one? Although in theory the "smaller" bubbles *rise* to the top, in reality it is the "heavier" or larger numbers which *sink* to the bottom. Remember, each smaller number can move upward only one tier for each pass, whereas larger numbers can fall several tiers in a single pass.**

FIRST PASS:

INITIAL VALUE					
8Ø	___	___	___	___	___
2Ø	___	___	___	___	___
35	___	___	___	___	___
12	___	___	___	___	___
1Ø	___	___	___	___	___
1	___	___	___	___	___

SECOND PASS:

INITIAL VALUE			
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___

THIRD PASS:

INITIAL VALUE			
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___
___	___	___	___

FOURTH PASS:

INITIAL VALUE		
___	___	___
___	___	___
___	___	___
___	___	___
___	___	___
___	___	___

FIFTH PASS:

INITIAL VALUE	
___	___
___	___
___	___
___	___
___	___
___	___

(Answers on Pages 32 and 33)

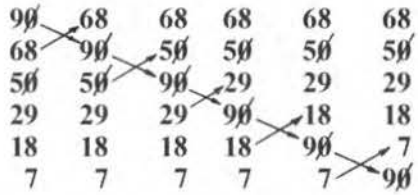
PROGRAMMER'S CHECK ANSWERS

2

1.

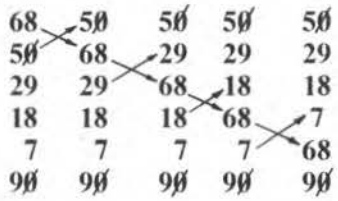
FIRST PASS:

INITIAL VALUE



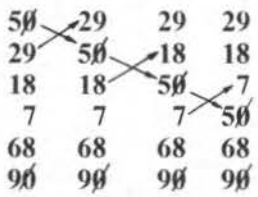
SECOND PASS:

INITIAL VALUE



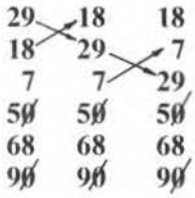
THIRD PASS:

INITIAL VALUE



FOURTH PASS:

INITIAL VALUE



FIFTH PASS:

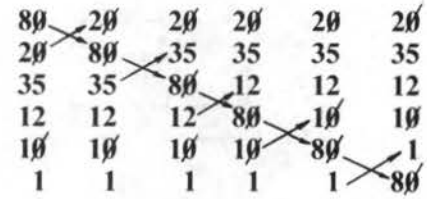
INITIAL VALUE

Programmer's Check 2 Answer (continued)

2.

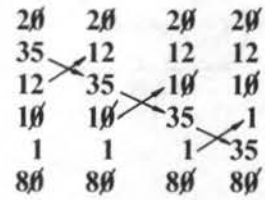
FIRST PASS:

INITIAL VALUE



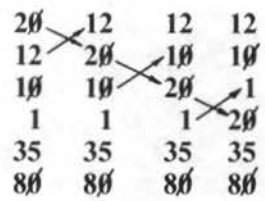
SECOND PASS:

INITIAL VALUE



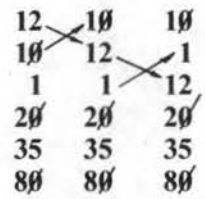
THIRD PASS:

INITIAL VALUE



FOURTH PASS:

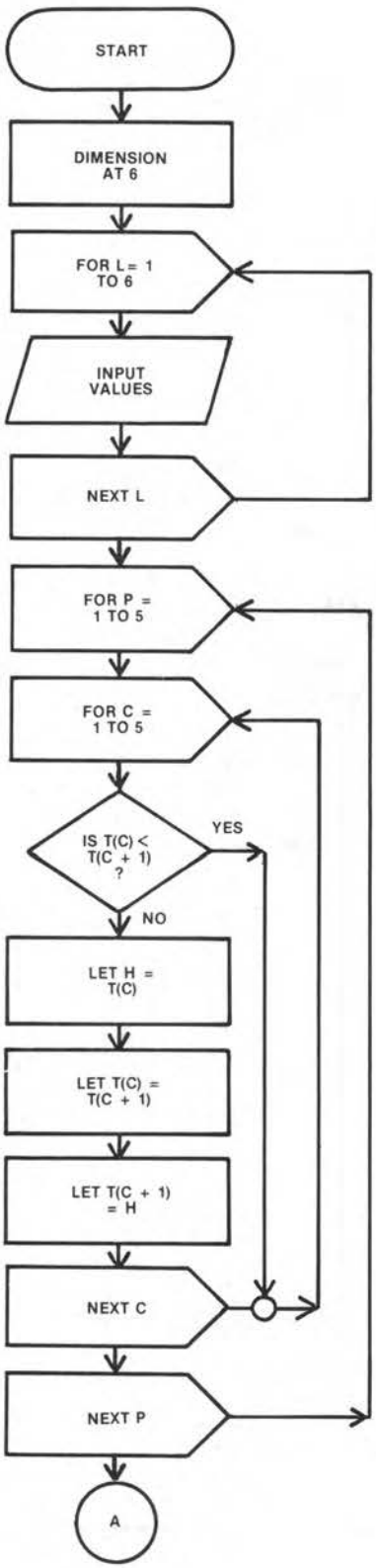
INITIAL VALUE

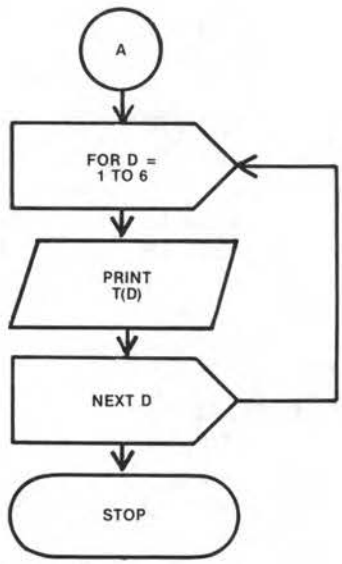


FIFTH PASS:

INITIAL VALUE

Now, let's look at the flowcharting design for a bubble sort:





Note how this design requires four FOR . . . NEXT loops. One of them merely serves to load six values into a table, while another is used to print out these six values after they have been sorted.

The other two loops are "nested"—that is, one is inside of the other. The FOR... NEXT loop controlled by the variable "P" will cause five passes through the table to occur.

The final loop controlled by "C" will compare one element T(C) with the next element in the table T(C + 1). If they are already in sequence, a branch will occur to "NEXT C". But, if not, the first value will be stored under the variable name "H". Then the second will be put where the first was. Finally, the value held in "H" will be put back into the table where the second value was. This is how the switching will occur.

Following is the coding for this bubble sort.

```
10 DIM T(6)
20 FOR L = 1 TO 6
30 INPUT T(L)
40 NEXT L
50 FOR P = 1 TO 5
60 FOR C = 1 TO 5
70 IF T(C) < T(C + 1) THEN GOTO
   110
80 LET H = T(C)
90 LET T(C) = T(C + 1)
100 LET T(C + 1) = H
110 NEXT C
120 NEXT P
130 FOR D = 1 TO 6
140 PRINT T(D)
150 NEXT D
160 STOP
```

ENTER this program and then RUN it. Use the values in the previous illustration and watch how the output will become sorted. Then, try running the program with other values. It should still work!

We can use the same method to sort character data as well! Imagine that we have a list of six names, none greater than eight characters long. Change line 10 to DIM T$(6,8). Then change every reference to "T" to T$. (You'll also have to change "H" to "H$".) Run the program as before, this time entering the following six names:

JOE

MARY

SAM

BARBARA

ALLEN

BETH

Once the run is complete, you will see them displayed as:

ALLEN

BARBARA

BETH

JOE

MARY

SAM

It would be just as simple a task to sort the records into descending sequence. All we would have to do is to change the comparison

on line 7Ø to test for the greater than (>) condition. Try doing just that.

```
7Ø IF T(C) > T(C + 1) THEN GOTO 11Ø
```

RUN the same program with the same names. This time, they should print as:

SAM

MARY

JOE

BETH

BARBARA

ALLEN

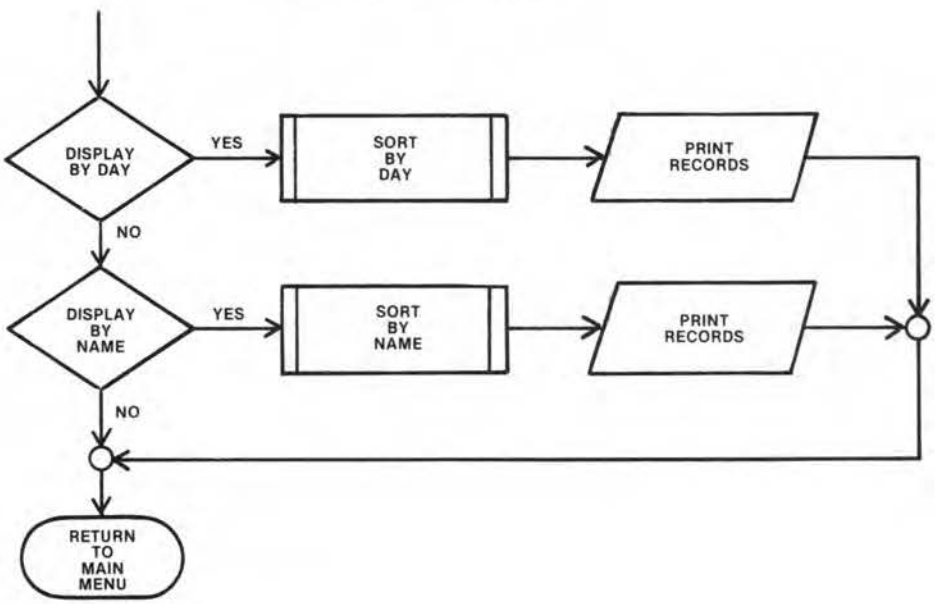
CODING THE DISPLAY MODULE

We will now use our knowledge of sorting to code the third and final module of our appointments calendar program.

When option three is chosen from the main menu, this module will be branched to. At this point we must ask the user whether they want records displayed by day or by name. Depending on which option they select, we will have to sort the records into the corresponding ascending sequence. We'll then print out the records slowly, one at a time.

The general logic to accomplish these tasks follows:

DISPLAY MODULE



The actual BASIC subroutines follow:

```
6000 REM DISPLAY SUBROUTINE
6010 CLS
6020 PRINT AT 3,0; "DISPLAY BY
     DAY OR NAME?"
6030 PRINT AT 5,0; "1 = BY DAY"
6040 PRINT AT 7,0; "2 = BY NAME"
6050 PRINT AT 11,0; "ENTER
     NUMBER OF SELECTION"
6060 INPUT R$
6070 IF R$ = "1" THEN GOSUB 6100
6080 IF R$ = "2" THEN GOSUB 6400
6090 RETURN
6100 CLS
6110 FAST
6120 FOR P = 1 TO N - 1
6130 FOR C = 1 TO N - 1
6140 IF D(C) < D(C + 1) THEN GOTO
     6300
6150 LET H = D(C)
6160 LET D(C) = D(C + 1)
6170 LET S$ = T$(C)
6180 LET T$(C) = T$(C + 1)
6190 LET V$ = P$(C)
6200 LET P$(C) = P$(C + 1)
6210 LET W$ = N$(C)
6220 LET N$(C) = N$(C + 1)
6230 LET X$ = M$(C)
6240 LET M$(C) = M$(C + 1)
6250 LET D(C + 1) = H
6260 LET T$(C + 1) = S$
6270 LET P$(C + 1) = V$
6280 LET N$(C + 1) = W$
6290 LET M$(C + 1) = X$
6300 NEXT C
6310 NEXT P
6320 SLOW
6330 GOSUB 6700
6340 RETURN
6400 CLS
6410 FAST
6420 FOR P = 1 TO N - 1
6430 FOR C = 1 TO N - 1
6440 IF N$(C) < N$(C + 1) THEN
     GOTO 6600
6450 LET W$ = N$(C)
6460 LET N$(C) = N$(C + 1)
6470 LET H = D(C)
6480 LET D(C) = D(C + 1)
6490 LET S$ = T$(C)
6500 LET T$(C) = T$(C + 1)
6510 LET V$ = P$(C)
6520 LET P$(C) = P$(C + 1)
6530 LET X$ = M$(C)
6540 LET M$(C) = M$(C + 1)
```

```
6550 LET N$(C + 1) = W$
6560 LET D(C + 1) = H
6570 LET T$(C + 1) = S$
6580 LET P$(C + 1) = V$
6590 LET M$(C + 1) = X$
6600 NEXT C
6610 NEXT P
6620 SLOW
6630 GOSUB 6700
6640 RETURN

6700 REM DISPLAY SORTED
     RECORDS

6710 FOR B = 1 TO N
6720 PRINT AT 1,0; "DAY "; D(B)
6730 PRINT AT 3,0; "TIME "; T$(B)
6740 PRINT AT 5,0; "PLACE "; P$(B)
6750 PRINT AT 7,0; "NAME "; N$(B)
6760 PRINT AT 9,0; "NOTES "; M$(B)
6770 PRINT AT 21,0; "PRESS ENTER
     TO CONTINUE"

6780 PAUSE 32767
6790 CLS
6800 NEXT B
6810 RETURN

8000 REM DISPLAY MENU

8010 CLS
8020 PRINT AT 0,4; "APPOINTMENTS
     CALENDAR"

8030 PRINT AT 2,0; "NUMBER"; AT
     2,16; "FUNCTION"

8040 PRINT AT 4,3; "1"; AT 4,12;
     "ENTER APPOINTMENTS"; AT
     5,13; "FOR A MONTH"

8050 PRINT AT 7,3; "2"; AT 7,12;
     "CHANGE OR REMOVE"; AT
     8,13; "APPOINTMENTS"

8060 PRINT AT 10,3; "3"; AT 10,12;
     "DISPLAY APPOINTMENTS"; AT
     11,13; "BY DAY OR NAME"

8070 PRINT AT 13,3; "4"; AT 13,12;
     "END THE PROGRAM"

8080 PRINT AT 21,0; "ENTER THE
     NUMBER OF YOUR CHOICE"

8090 INPUT A$
8100 IF A$ = "1" OR A$ = "2" OR A$
     = "3" OR A$ = "4" THEN
     GOTO 8140

8110 PRINT AT 15,3; A$; AT 15,6; "IS
     AN INVALID SELECTION"; AT
     16,4; "PRESS ENTER AND TRY
     AGAIN"

8120 PAUSE 32767
8130 GOTO 8000
8140 IF A$ = "1" THEN GOSUB 1000
8150 IF A$ = "2" THEN GOSUB 3000
8160 IF A$ = "3" THEN GOSUB 6000
8170 IF A$ = "4" THEN RETURN
8180 GOTO 8000
```

Note that when the appropriate choice is made, the records are sorted into the proper sequence. Only the name or the day is used for the key field. When a record is found out of sequence, the key field must be switched along with all of the other values (place, time, and notes).

Also, since "N" contains the number of records in the tables, this value is used to control the number of passes and comparisons which have to be made in the sort (N – 1) and the number of records to be displayed.

One more addition must be made for our program to be complete. We have to suppress the printing of logically deleted records. This can be accomplished simply by adding one statement to our print logic:

```
6715 IF D(B) = Ø THEN GOTO 68ØØ
```

Insert this coding into the previously saved program. Now that all of the modules have been completed, the entire program should run without error. You may wish to actually use this program to store your own appointments. You now have a piece of software that would have cost between $10 and $15 if purchased from a computer store!

Now, see what you have learned by taking the Programmer's Check which follows. Maybe you can think of other applications which you could create. Perhaps they are marketable! At the very least, you should be able to modify purchased software to suit your own needs; this is known as customizing.

FIGURE 10—Getting along with others and being able to communicate effectively in a business environment are as basic to your success as BASIC.

PROGRAMMER'S CHECK

3

Adding Program Modules

1. **Frequently, the programmer is requested by a client to modify an existing program to meet new user requirements. Here is an example of how new modules can be added to a program you already have on tape.**

Program Name: **IU1ØA2 (Instruction Unit 1Ø, Assignment 2)**

Type: **MENU-DRIVEN RECORD LIBRARY**

Specifications:

Design, code and debug additions to the record library program developed earlier in this Study Unit. Step "D" will require you to bubble sort the file by either artist or type.

Add the following modules:

A. A module to add new records.

B. A module to delete old albums.

C. A module to make changes to old albums.

D. A module to display albums by artist or type.

2. **After you have debugged this program, you might consider offering the use of it to friends and relatives. In any case, you should now attempt to teach at least one person how to use this program. See whether you have sufficient menu "helps". Begin to understand what "user problems" occur by observing how others require guidance and instruction. Then, modify the program as necessary to meet user requirements.**

DO YOU KNOW NOW?

These were the questions posed at the beginning of the lesson.

- **How menus can make software "user-friendly"?**
  Menus are printed displays of the options which the user of the system can choose from. A user-driven program requires that the user merely read the printed instructions. All of the intricacies of the program are kept inside of the program and away from the user.

- **How a bubble sort works?**
  A bubble sort floats the lesser values in a table to the top of the table. Each value is compared to the next value and the elements are switched if they are out of sequence; otherwise, they are left alone. Bubble sorts can put tables composed of numbers or characters into either ascending or descending sequence.

- **What stub testing means?**
  Stub testing refers to the process of program development which allows for a program to be designed, coded and tested without the need for entering the details of the entire program at once. Stub modules are merely branched to and returned from in the early stages of development. Later on, they are coded in full.

SCHOOL OF COMPUTER TRAINING

EXAM 10

MERGING — FUNCTIONS

24710-2

Questions 1-10: *Circle the letter beside the one best answer to each question*

1. In data processing, a "stub" is

   (a) the label of a program.
   (b) an uncoded module.
   (c) a convenient way to test part of a program.
   (d) a GOSUB menu.

2. A menu-driven program

   (a) requires technical knowledge of the user.
   (b) helps to make a user-friendly program.
   (c) ensures that the program will run.
   (d) saves the program onto cassette tape.

3. User-friendly software

   (a) requires no technical knowledge by the user.
   (b) allows the user to "change his mind".
   (c) makes use of menus.
   (d) all of the above.

4. Logical deletion

   (a) purges records from a file.
   (b) flags records as deleted.
   (c) set the numeric variables to blanks.
   (d) set the character variables to zero.

5. A sort

   (a) rearranges every record in a table.
   (b) can be ascending or descending.
   (c) can be numeric only.
   (d) can be non-numeric only.

6. A bubble sort is so called because

   (a) nested FOR...NEXT loops are used.
   (b) bubble memory is necessary.
   (c) the lesser values float toward the top after each pass.
   (d) none of the above.

7. In a bubble sort

   (a) the number of passes must be equal to the number of elements in the table.
   (b) the number of passes must be one *less* than the number of elements in the table.
   (c) the number of passes must be one *more* than the number of elements in the table.
   (d) there is no relationship between the number of passes and the number of elements in the table.

8. When we are to change or delete a record from a file

   (a) the record to be changed does not have to be found.
   (b) the record to be deleted must be physically removed from the file.
   (c) we can just add a new, changed record to the file.
   (d) the table must first be searched.

9. Merging is the process of

   (a) putting two files together.
   (b) sorting records according to numerical value.
   (c) sorting records according to alpha value.
   (d) arranging files in descending order.

10. "Nesting" refers to

   (a) two or more loops being in the same program.
   (b) a FOR...NEXT loop.
   (c) one loop within another loop.
   (d) the position of a chip in the computer.

WHEN YOU HAVE COMPLETED THE ENTIRE EXAM, TRANSFER YOUR ANSWERS TO THE ANSWER SHEET WHICH FOLLOWS.

ICS

ANSWER PAPER

To avoid delay, please insert all the details requested below

Subject _____ Course _____

Name

Address

Post Code

Serial Number: 2 4 7 1 0

Test No.: 10

Ed No.: 2

Student's Reference

Letters / Figures

Tutor's Comments

Grade	Tutor

Study the foregoing Question Paper and use it for your rough workings. Record your final answers in the matrix below by writing a cross (X), IN INK OR BALLPOINT, through the letter which you think is the correct answer. Submit ONLY THIS ANSWER SHEET to the School for correction. ALL QUESTIONS MUST BE ANSWERED.

1.	A	B	C	D
2.	A	B	C	D
3.	A	B	C	D
4.	A	B	C	D
5.	A	B	C	D

6.	A	B	C	D
7.	A	B	C	D
8.	A	B	C	D
9.	A	B	C	D
10.	A	B	C	D